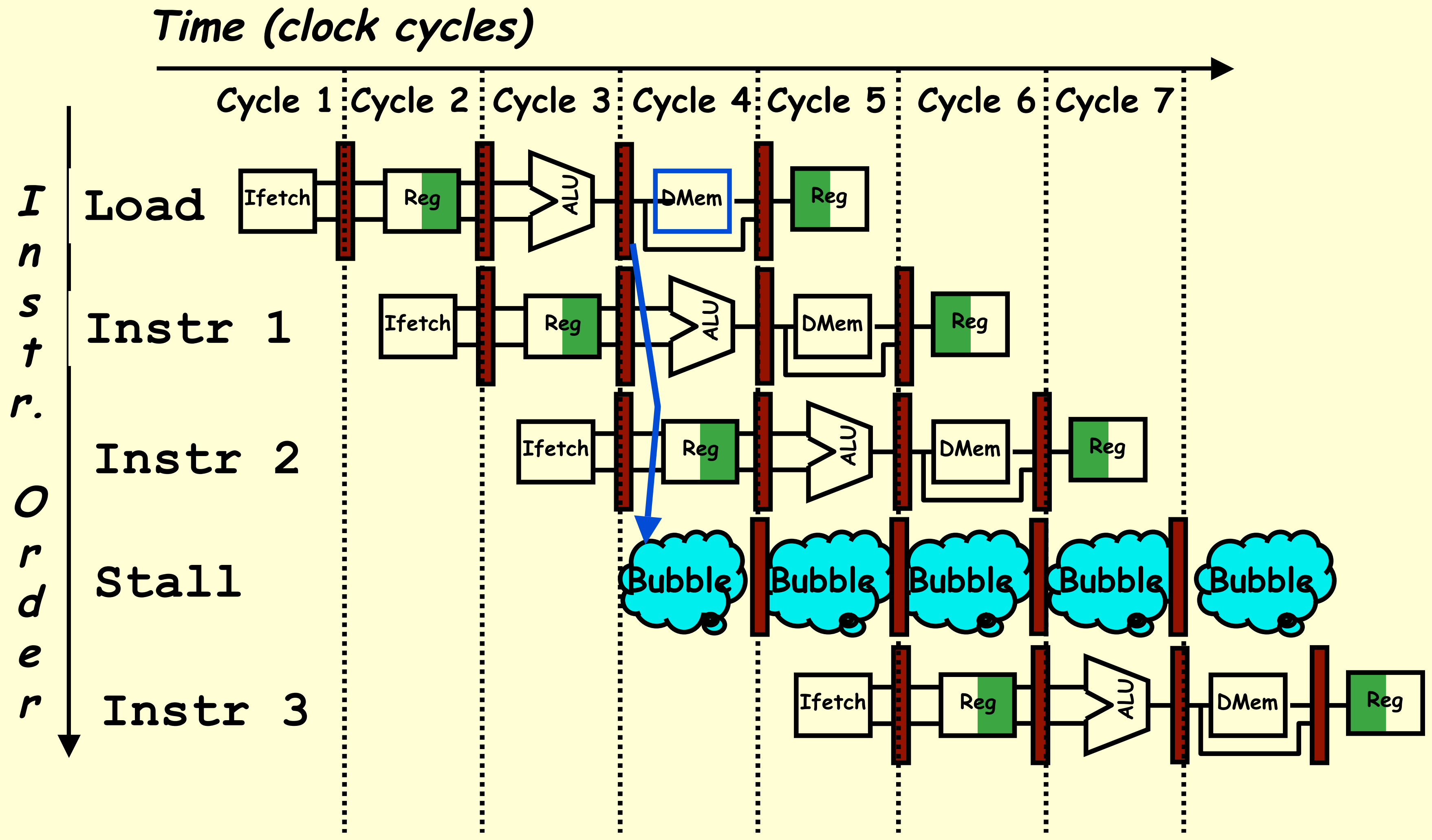
CMSC 611: Advanced Computer Architecture

Pipelining

Pipeline Hazards

- Cases that affect instruction execution semantics and thus need to be detected and corrected
- Hazards types
  - Structural hazard: attempt to use a resource two different ways at same time
    - Single memory for instruction and data
  - Data hazard: attempt to use item before it is ready
    - Instruction depends on result of prior instruction still in the pipeline
  - Control hazard: attempt to make a decision before condition is evaluated
    - branch instructions
- Hazards can always be resolved by waiting

Detecting and Resolving Structural Hazard



Slide: David Culler

Stalls & Pipeline Performance

$$\text{Pipelining Speedup} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

$$\text{Ideal CPI pipelined} = 1$$

$$\text{CPI pipelined} = \text{Ideal CPI} + \text{Pipeline stall cycles per instruction}$$

$$= 1 + \text{Pipeline stall cycles per instruction}$$

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

Assuming all pipeline stages are balanced

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

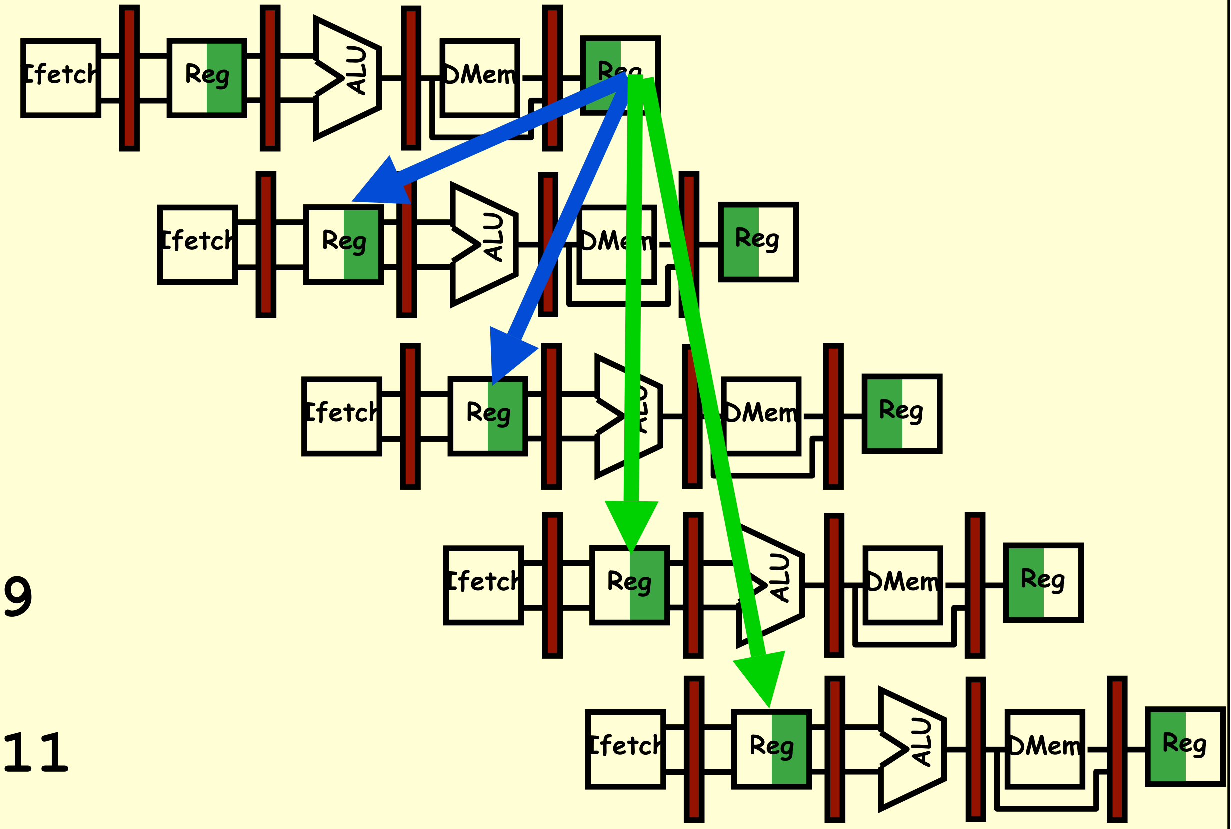
Data Hazards

Time (clock cycles)

IF ID/RF EX MEM WB

Instr. Order

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
or   r8,r1,r9
xor r10,r1,r11
```



Slide: David Culler

Three Generic Data Hazards

- Read After Write (RAW)
  Instr_J tries to read operand before Instr_I writes it

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- Caused by a "Data Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Three Generic Data Hazards

- Write After Read (WAR)
  Instr$_J$ writes operand before Instr$_I$ reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "anti-dependence" in compilers.
  - This results from reuse of the name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Slide: David Culler

Three Generic Data Hazards

- Write After Write (WAW)
  Instr$_J$ writes operand before Instr$_I$ writes it.

```
I: mul r1,r4,r3
J: add r1,r2,r3
K: sub r6,r1,r7
```

- Called an "output dependence" in compilers
  - This also results from the reuse of name "r1".
- Can't happen in MIPS 5 stage pipeline:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Do see WAR and WAW in more complicated pipes

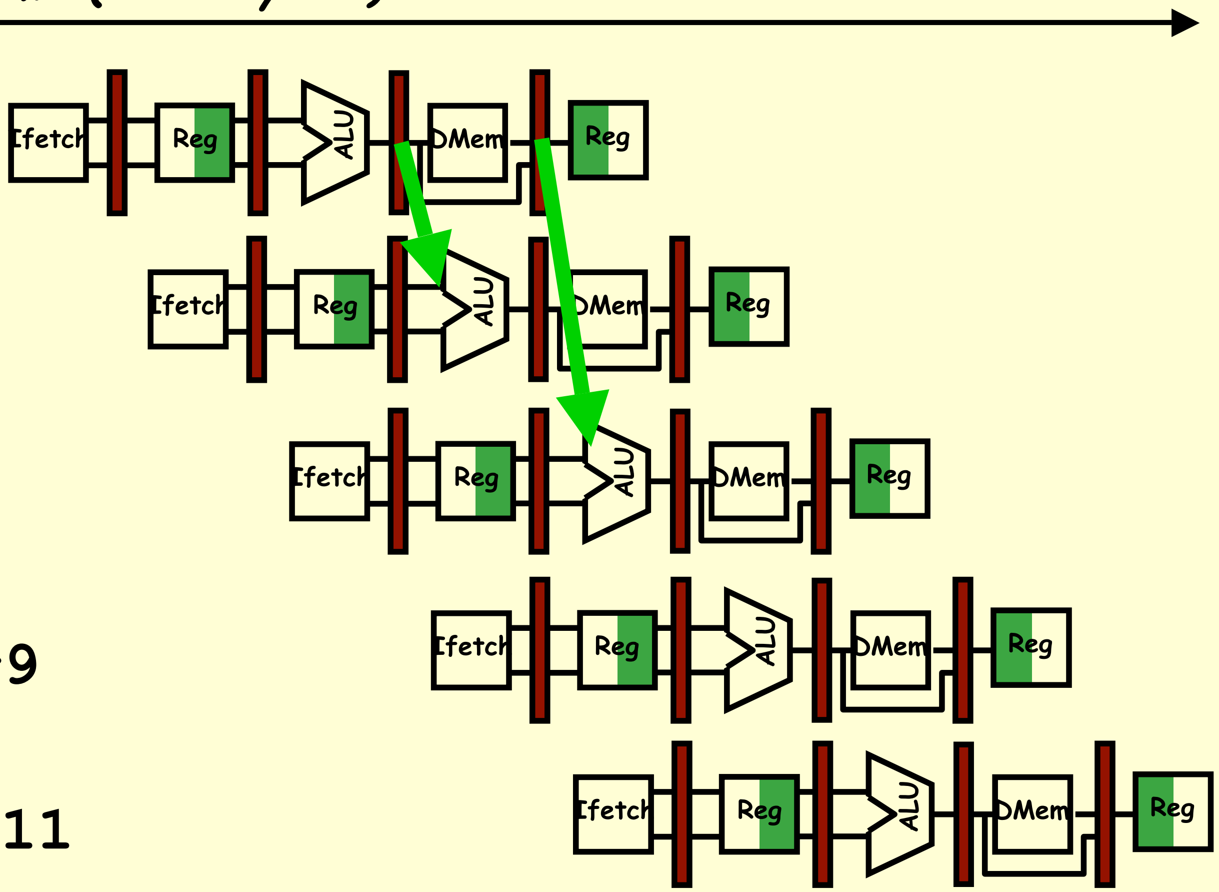
Slide: David Culler

Forwarding to Avoid Data Hazard

Time (clock cycles)

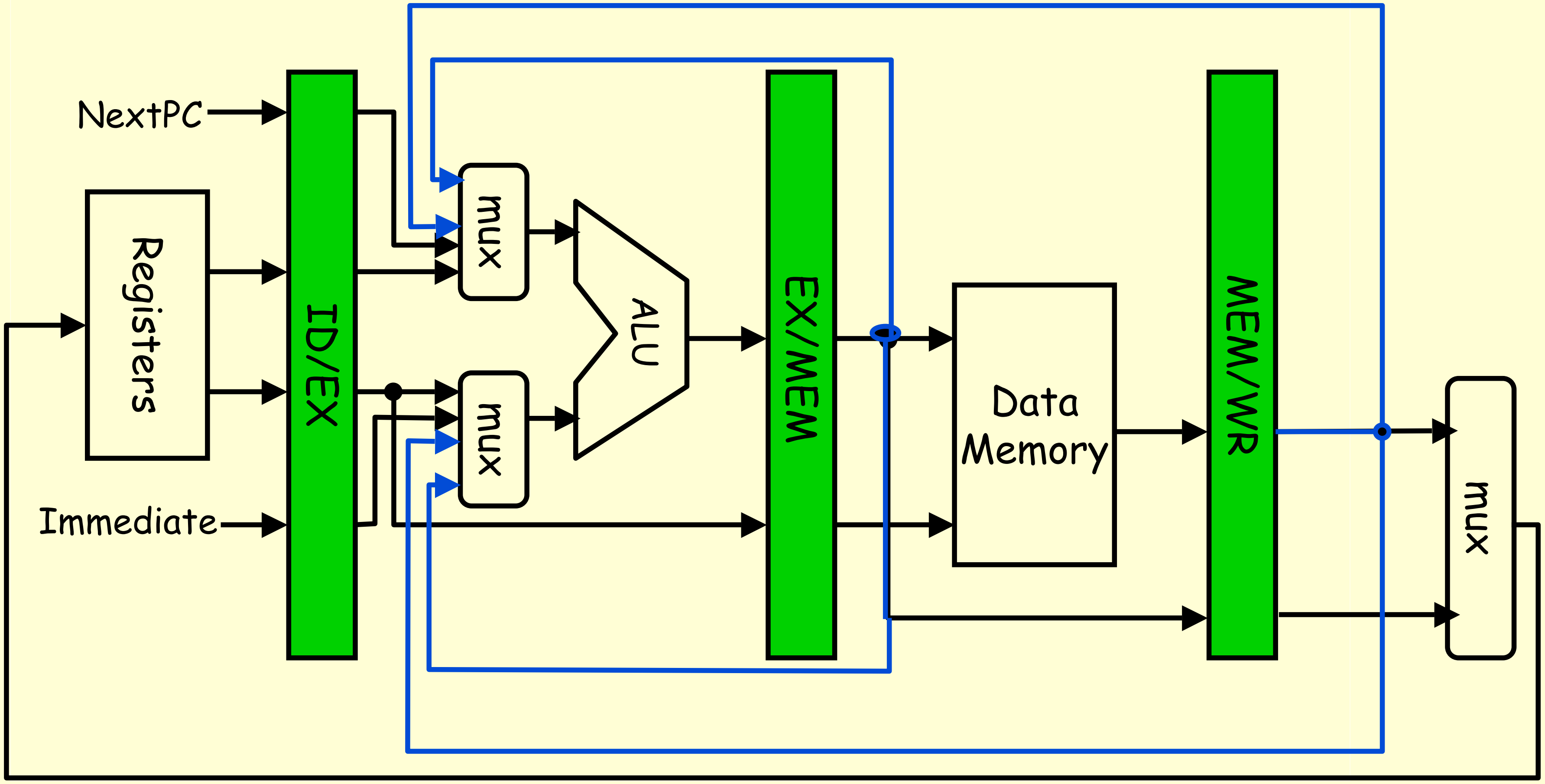
Instr. Order

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
or   r8,r1,r9
xor r10,r1,r11
```



Slide: David Culler

HW Change for Forwarding



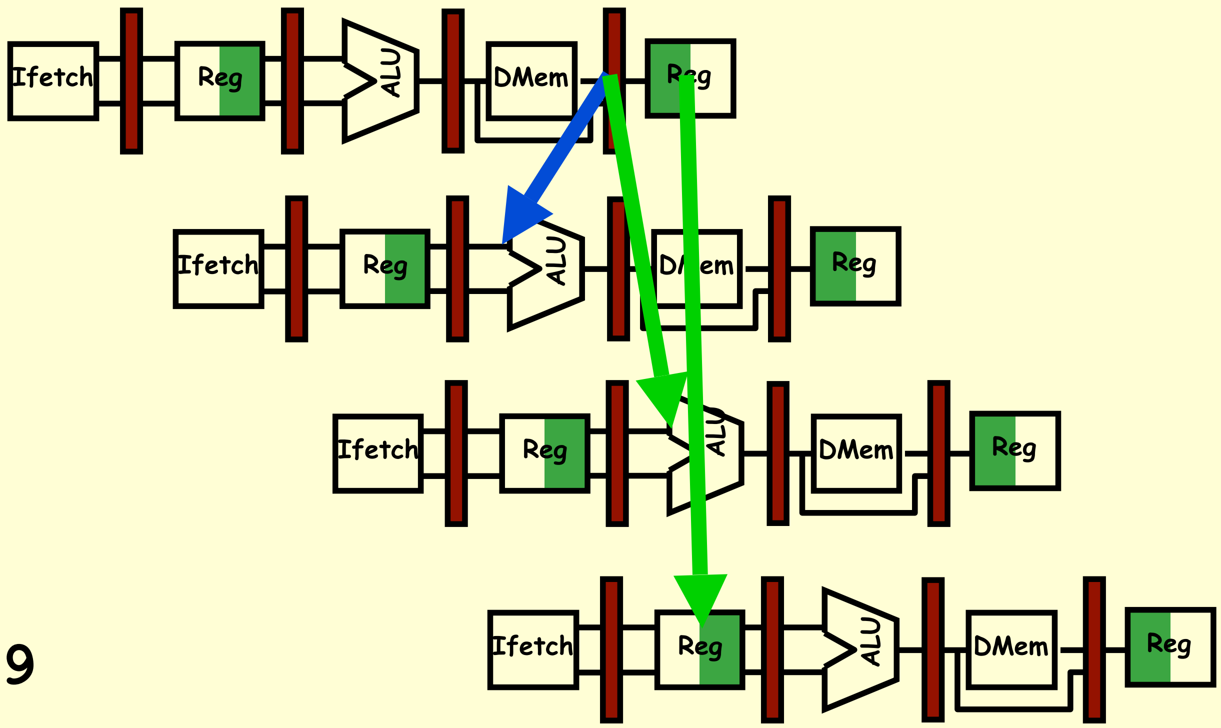
Slide: David Culler

Data Hazard Even with Forwarding

Time (clock cycles)

Instr. Order

```
lw r1, 0(r2)
sub r4,r1,r6
and r6,r1,r7
or   r8,r1,r9
```



Slide: David Culler

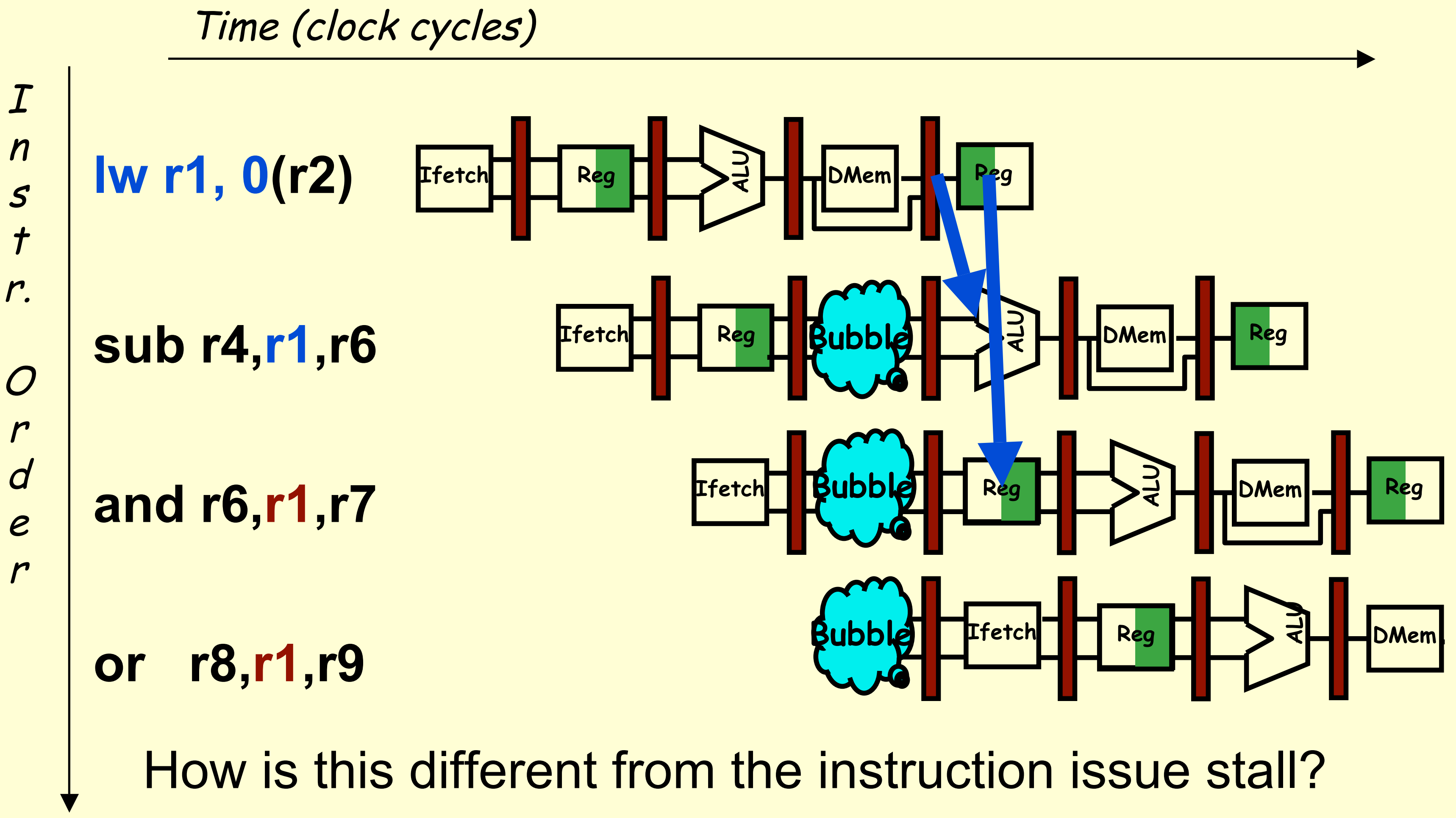
Resolving Load Hazards

- Adding hardware? How? Where?
- Detection?
- Compilation techniques?

- What is the cost of load delays?

Slide: David Culler

Resolving the Load Data Hazard



How is this different from the instruction issue stall?

Slide: David Culler

Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

Slow code:

```
LW      Rb,b
LW      Rc,c
ADD     Ra,Rb,Rc
SW      a,Ra
LW      Re,e
LW      Rf,f
SUB     Rd,Re,Rf
SW      d,Rd
```

Fast code:

```
LW      Rb,b
LW      Rc,c
LW      Re,e
ADD     Ra,Rb,Rc
LW      Rf,f
SW      a,Ra
SUB     Rd,Re,Rf
SW      d,Rd
```

Slide: David Culler

Instruction Set Connection

- What is exposed about this organizational hazard in the instruction set?
- k cycle delay?
  - bad, CPI is not part of ISA
- k instruction slot delay
  - load should not be followed by use of the value in the next k instructions
- Nothing, but code can reduce run-time delays
- MIPS did the transformation in the assembler

Slide: David Culler