Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Data Hazard 1

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (μpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- Data Hazard

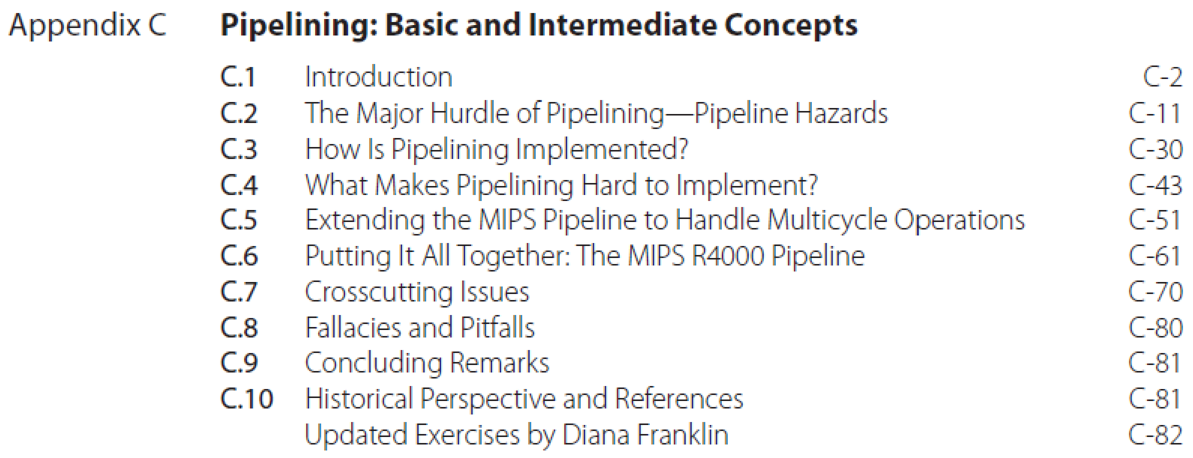
  RAW, WAR, WAW Hazards

Unit 3: **Memory Design**
Unit 4: **Input/Output Device Design**
Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (μpCA)

Text 1: “Computer Organization and Design”, Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:“Computer Architecture: A Quantitative Approach”, Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.



Data Hazard

Consider a Scenario

Suppose initially, register ***i*** holds the number 2i
(ie. R3 = 6; R6=12; R8 = 16…)

What happens when the following sequence is executed in 5-stage pipeline?

add R3, R10, R11 - this should add 20 + 22, putting result 42 into r3

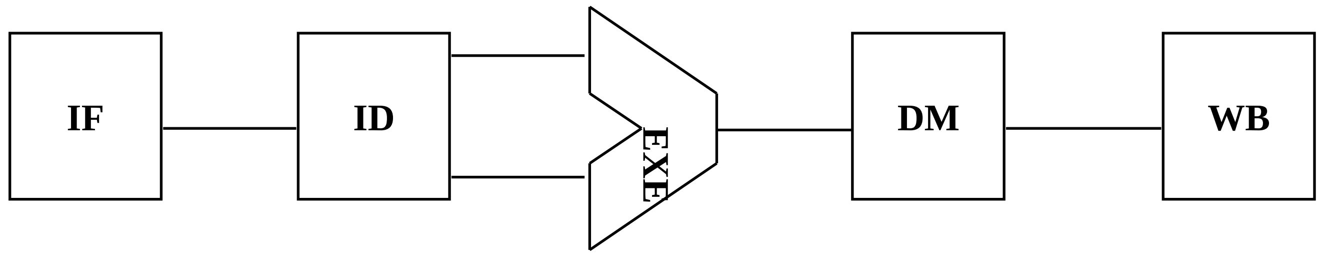
ldr R8, [R3, #50] - this should load r8 from memory location 42+50 = 92

sub R11, R8, R7 - this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (μpCA)

Cycle 1

add R3, R10, R11



IF ID EX MEM WB

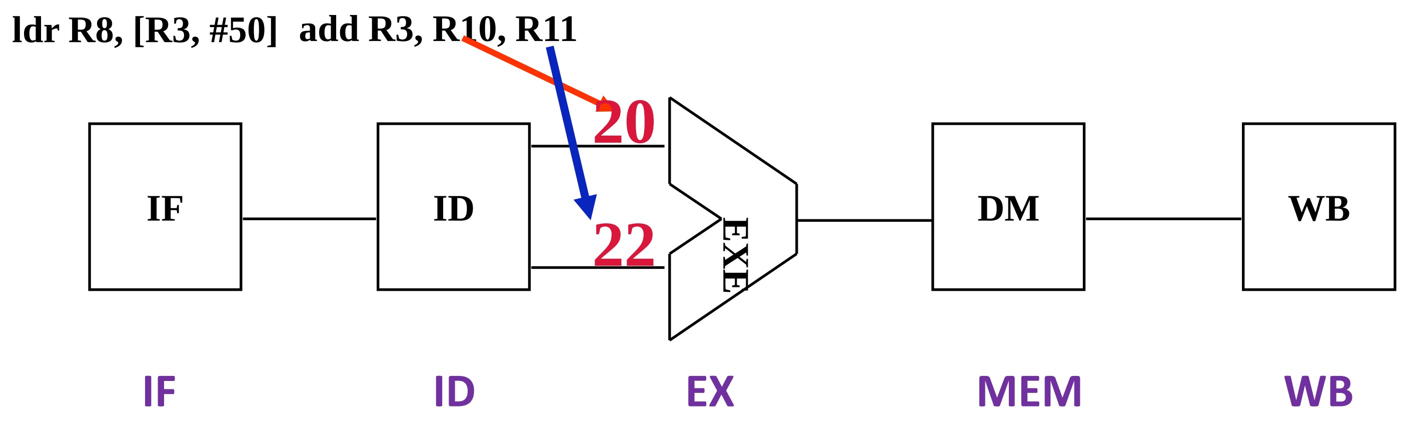
add **R3, R10, R11** - this should add 20 + 22, putting result 42 into r3

ldr **R8, [R3, #50]** - this should load r8 from memory location 42+50 = 92

sub **R11, R8, R7** - this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (µpCA)

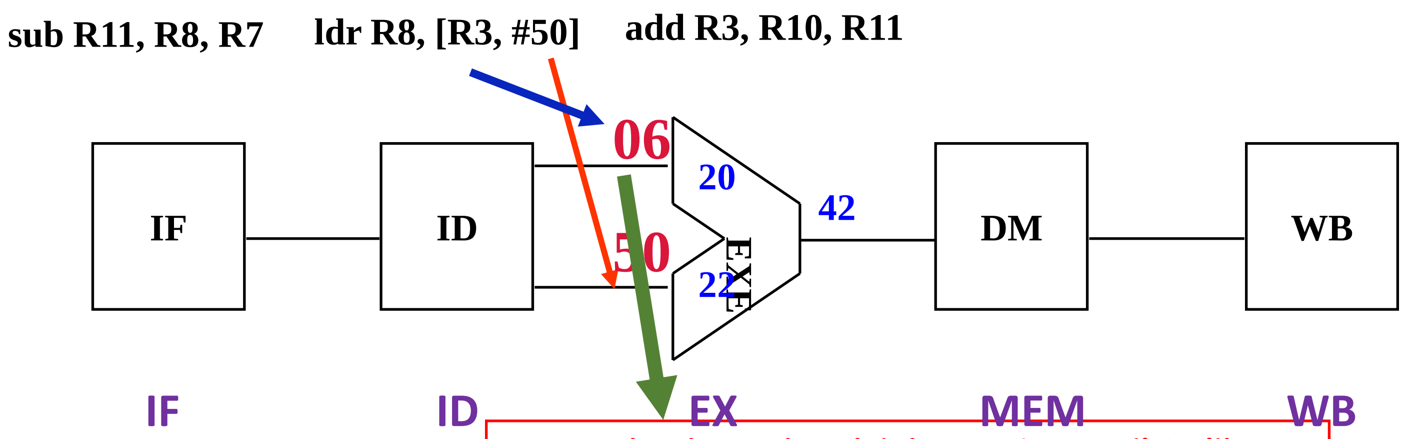
Cycle 2



add	**R3, R10, R11**	- this should add 20 + 22, putting result 42 into r3
ldr	**R8, [R3, #50]**	- this should load r8 from memory location 42+50 = 92
sub	**R11, R8, R7**	- this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (µpCA)

Cycle 3

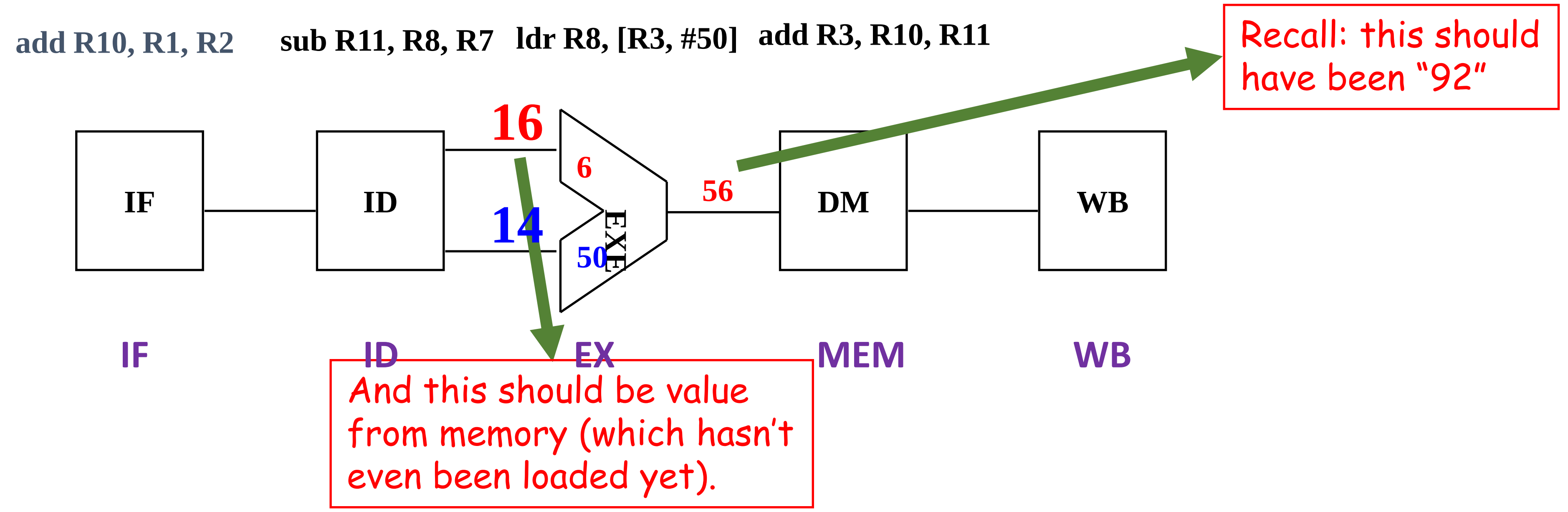


Ooops! This should have been "42"!
But register 3 didn't get updated yet.

add	**R3, R10, R11**	- this should add 20 + 22, putting result 42 into r3
ldr	**R8, [R3, #50]**	- this should load r8 from memory location 42+50 = 92
sub	**R11, R8, R7**	- this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (μpCA)

Cycle 4



add R10, R1, R2
sub R11, R8, R7
ldr R8, [R3, #50]
add R3, R10, R11
Recall: this should have been "92"
IF
ID
16
6
14
50
EXE
56
DM
WB
IF
ID
EX
MEM
WB
And this should be value from memory (which hasn't even been loaded yet).

add	**R3, R10, R11**	- this should add 20 + 22, putting result 42 into r3
ldr	**R8, [R3, #50]**	- this should load r8 from memory location 42+50 = 92
sub	**R11, R8, R7**	- this should subtract 14 from that just-loaded value from Mem[92]

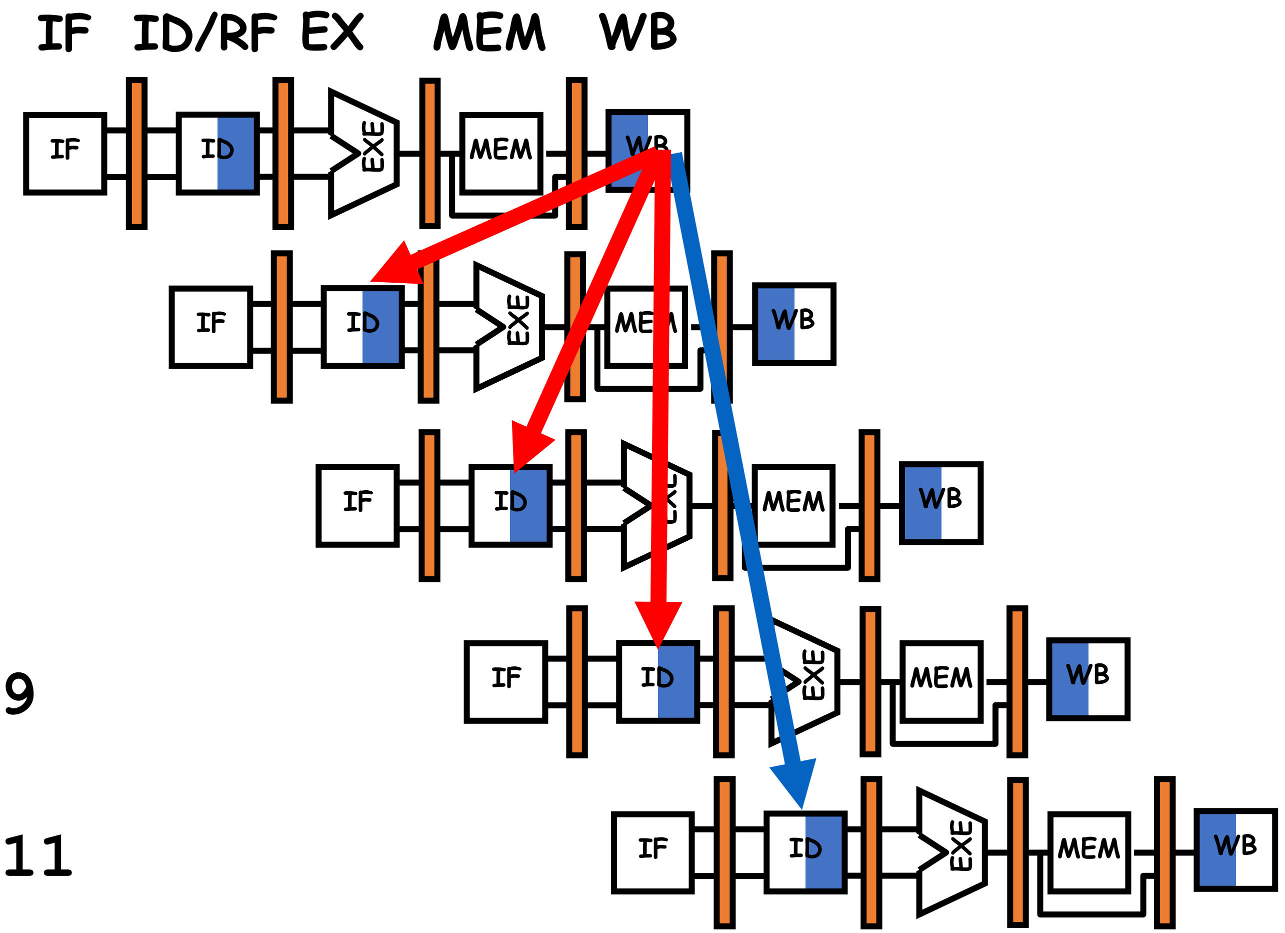
- Problem with starting next instruction before first is finished
  - Data dependencies here, that "go backward in time" create data hazards.

Microprocessor & Computer Architecture (μpCA)

Data Dependency

Instr. Order

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



Data Dependencies

- Three types of data dependencies defined in terms of how succeeding instruction depends on preceding instruction
  - **RAW:** Read after Write or Flow dependency
    (True Dependency)
  - **WAR:** Write after Read or anti-dependency
    (Anti Dependency)
  - **WAW:** Write after Write
    (Output Dependency)

Data Dependencies

- 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 “Dependence” (in compiler nomenclature). This hazard results from an actual need for communication.

Data Dependencies

- Example program (a) with two instructions
  - i1: load **r1,** a;
  - i2: add r2, **r1,**r1;
- Program (b) with two instructions
  - i1: mul **r1,** r4, r5;
  - i2: add r2, **r1,** r1;
- Both cases we cannot read in i2 until i1 has completed writing the result
  - In **(a)** this is due to *load-use dependency*
  - In **(b)** this is due to *define-use dependency*

Data Dependencies

- 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" by compiler writers.
  This results from reuse of the name "r1".

- Can't happen if in pipeline:-
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Data Dependencies

- Write After Write (WAW)
  Instr$_J$ writes operand *before* Instr$_I$ writes it.
- Called an “output dependence” by compiler writers
  This also results from the reuse of name “r1”.
- Can’t happen if in pipeline:
- All instructions take 5 stages, and Writes are always in stage 5

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

If Executed Out of order by the Compiler, may lead to wrong results

WAR and WAW Dependency

- Example program (a):
  - i1: mul r1, r2, r3;
  - i2: add r2, r4, r5;
- Example program (b):
  - i1: mul r1, r2, r3;
  - i2: add r1, r4, r5;
- both cases we have dependence between i1 and i2
  - in (a) due to r2 must be read before it is written into
  - in (b) due to r1 must be written by i2 after it has been written into by i1

WAR and WAW Dependency

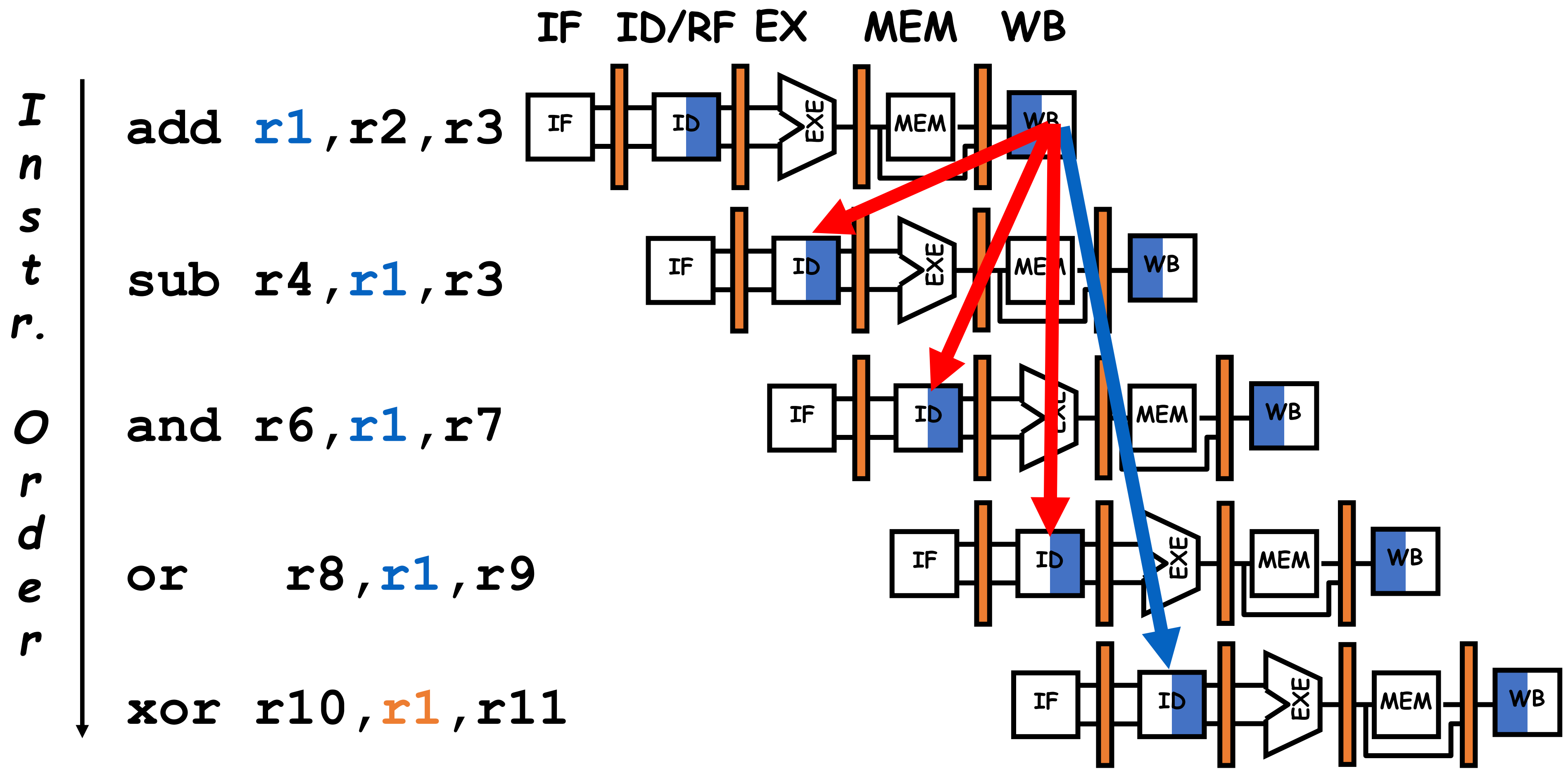
- Problem:
  - i1: mul r1, r2, r3;
  - i2: add r2, r4, r5;
- Is this really a dependence/hazard ?

WAR and WAW Dependency

- Solution: Rename Registers
  - i1: mul r1, r2, r3;
  - i2: add r6, r4, r5;
- Register renaming can solve many of these *false dependencies*
  - note the role that the compiler plays in this
  - specifically, the register allocation process--i.e., the process that assigns registers to variables

Microprocessor & Computer Architecture (μpCA)

Next Session: How to Attack RAW Dependency?



THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135