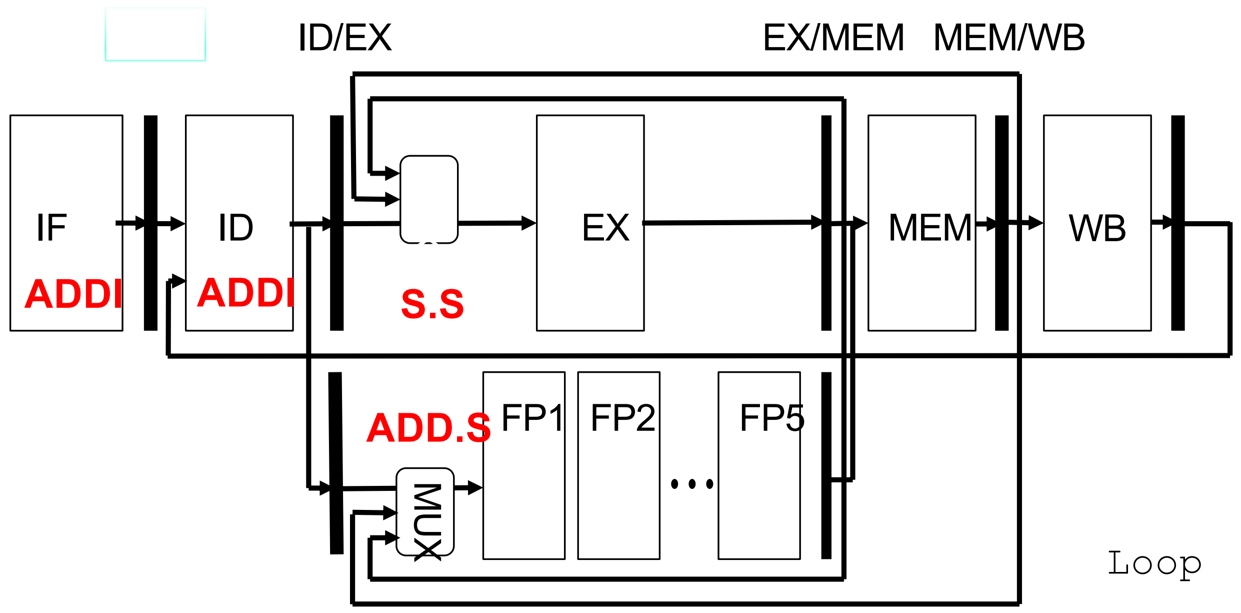
Lecture 3

Instruction scheduling techniques

- **Dynamically scheduled pipelines (Ch. 3.4)**
  - ✓ Tomasulo's algorithm (3.4.1)

Dynamic Instruction Scheduling

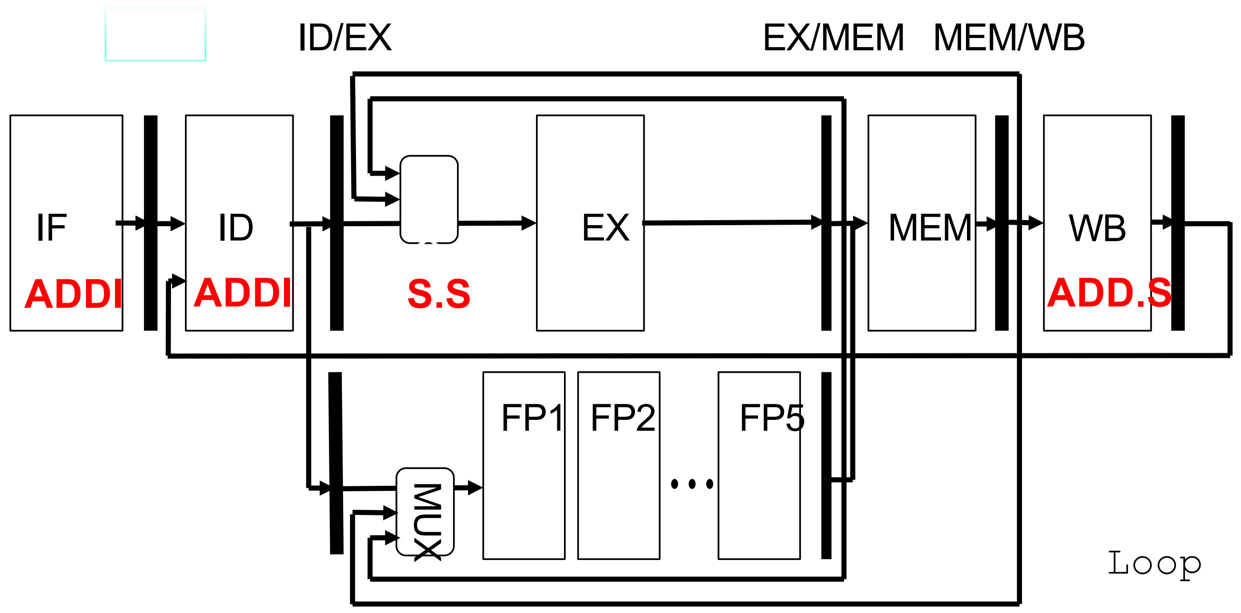
Limitations of Static Scheduling



```
Loop    L.S F0,0(R1)     (1)
        L.S F1,0(R2)     (1)
        ADD.S F2,F1,F0   (2)
        S.S F2,0(R1)     (5)
        ADDI R1,R1,#4    (1)
        ADDI R2,R2,#4    (1)
        SUBI R3,R3,#1    (1)
        BNEZ R3,Loop     (3)
```

Five cycles later

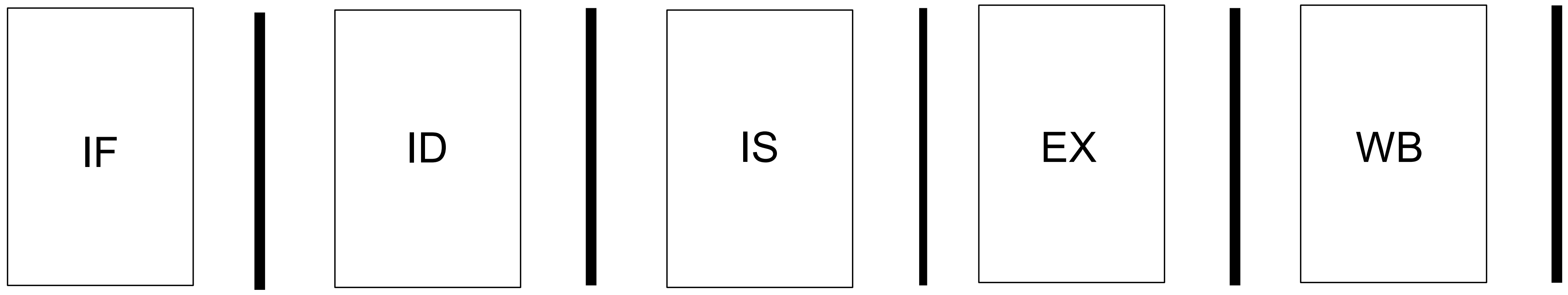
Limitations of Static Scheduling



```
Loop    L.S F0,0(R1)      (1)
        L.S F1,0(R2)      (1)
        ADD.S F2,F1,F0    (2)
        S.S F2,0(R1)      (5)
        ADDI R1,R1,#4     (1)
        ADDI R2,R2,#4     (1)
        SUBI R3,R3,#1     (1)
        BNEZ R3,Loop      (3)
```

Question:
How many cycles would the code need, ideally?

Answer:
One cycle per instruction yields eight cycles



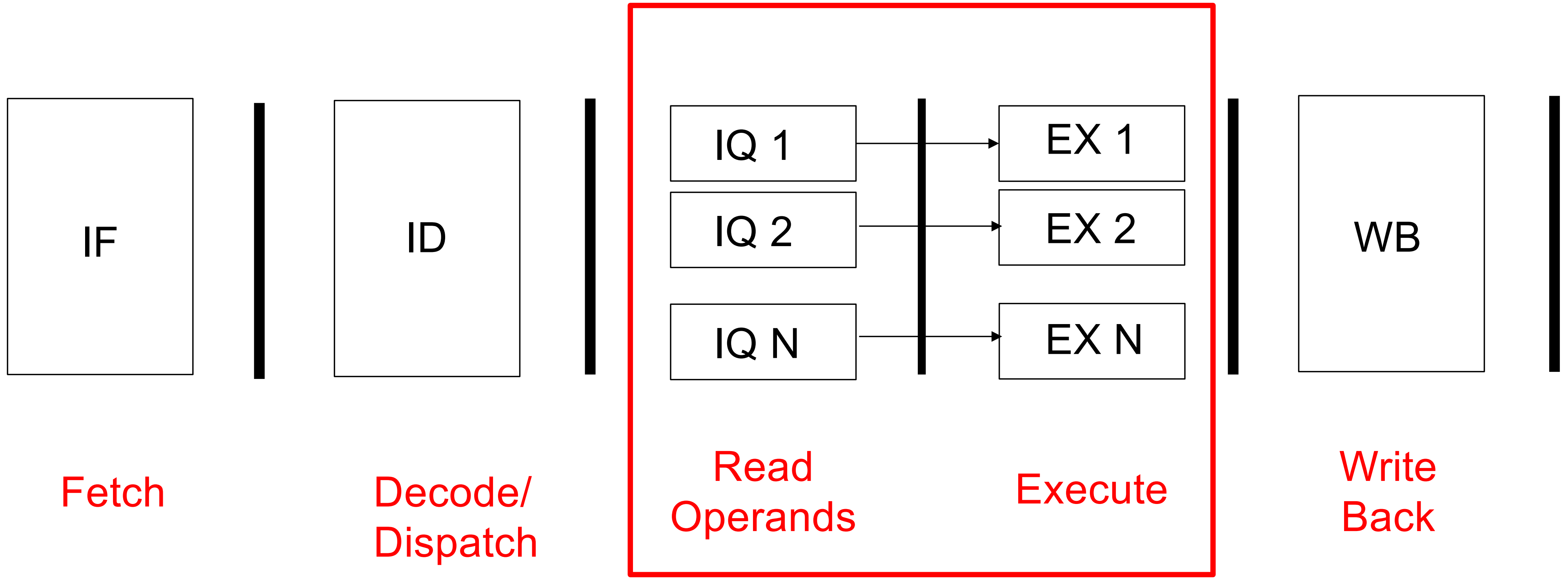
Instruction Fetch

Instruction Decode/ Dispatch

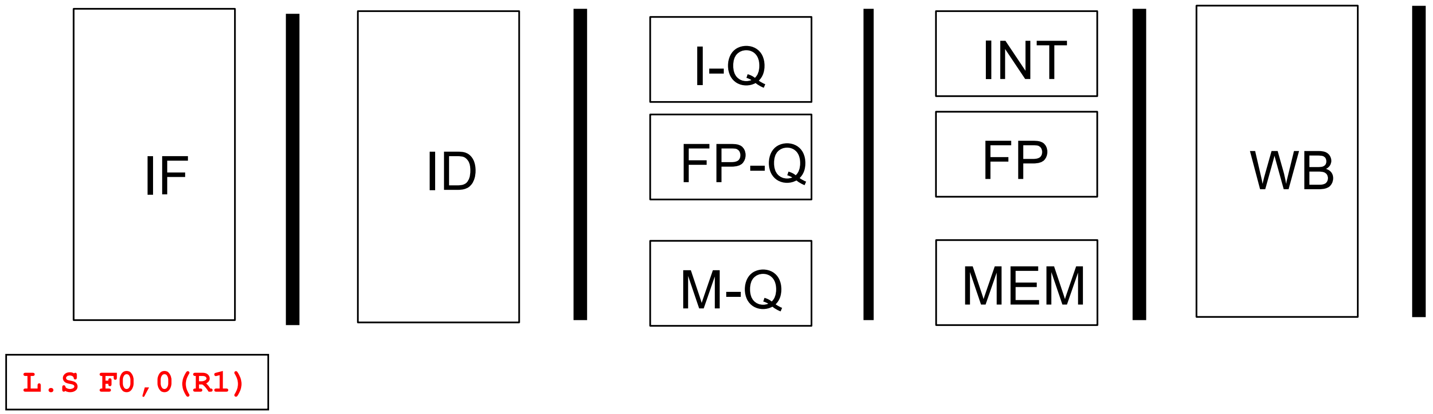
Instruction Issue/Read Operands

Execute

Write Back

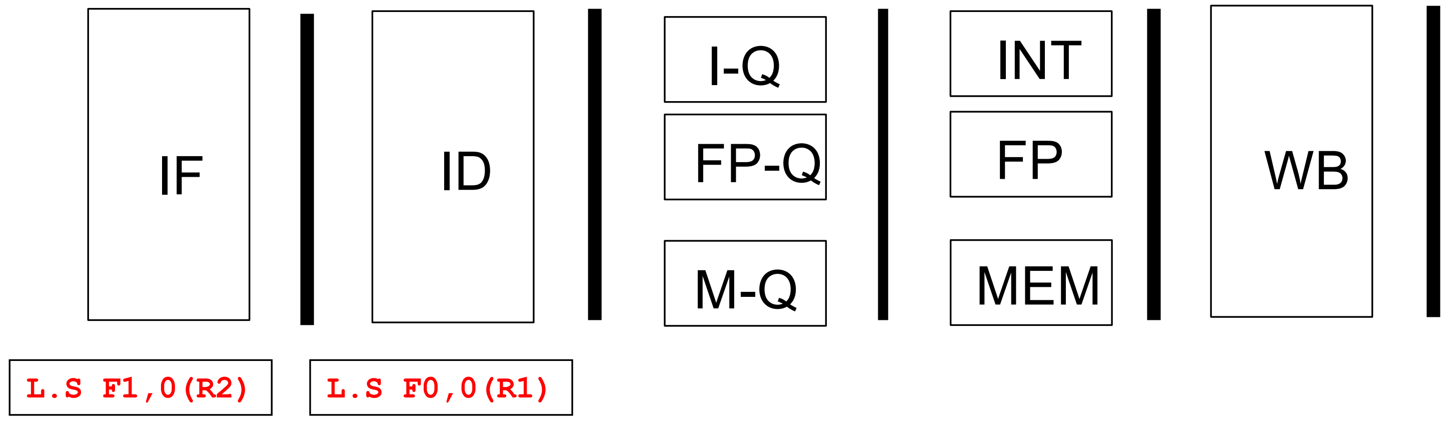

IF
ID
IQ 1
IQ 2
IQ N
EX 1
EX 2
EX N
WB
Fetch
Decode/ Dispatch
Read Operands
Execute
Write Back

Cycle: 1



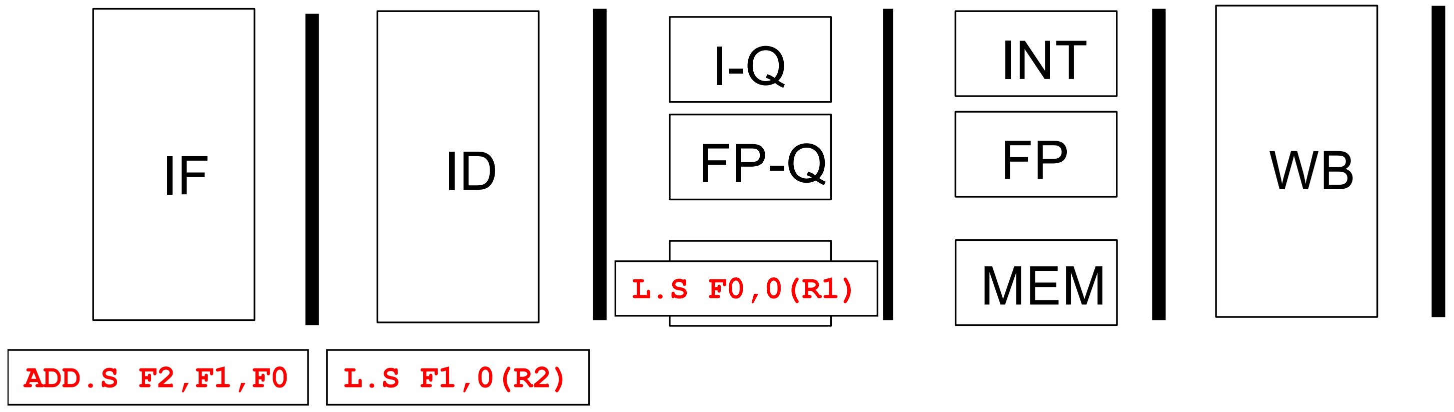
```
Loop ➡ L.S F0,0(R1)
       L.S F1,0(R2)
       ADD.S F2,F1,F0
       S.S F2,0(R1)
       ADDI R1,R1,#4
       ADDI R2,R2,#4
       SUBI R3,R3,#1
       BNEZ R3,Loop
```

Cycle: 2



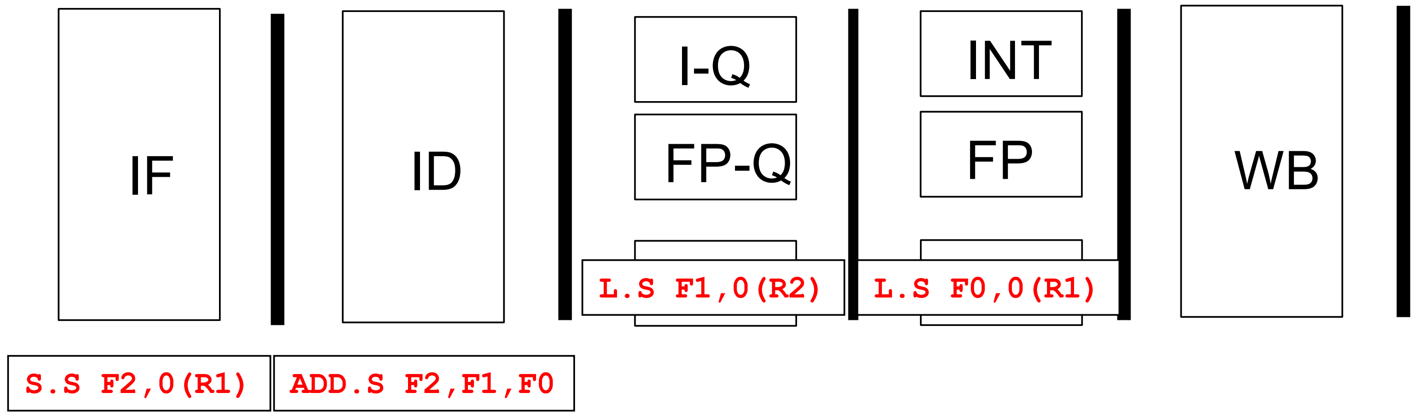
```
Loop    L.S F0,0(R1)
     ➡  L.S F1,0(R2)
        ADD.S F2,F1,F0
        S.S F2,0(R1)
        ADDI R1,R1,#4
        ADDI R2,R2,#4
        SUBI R3,R3,#1
        BNEZ R3,Loop
```

Cycle: 3

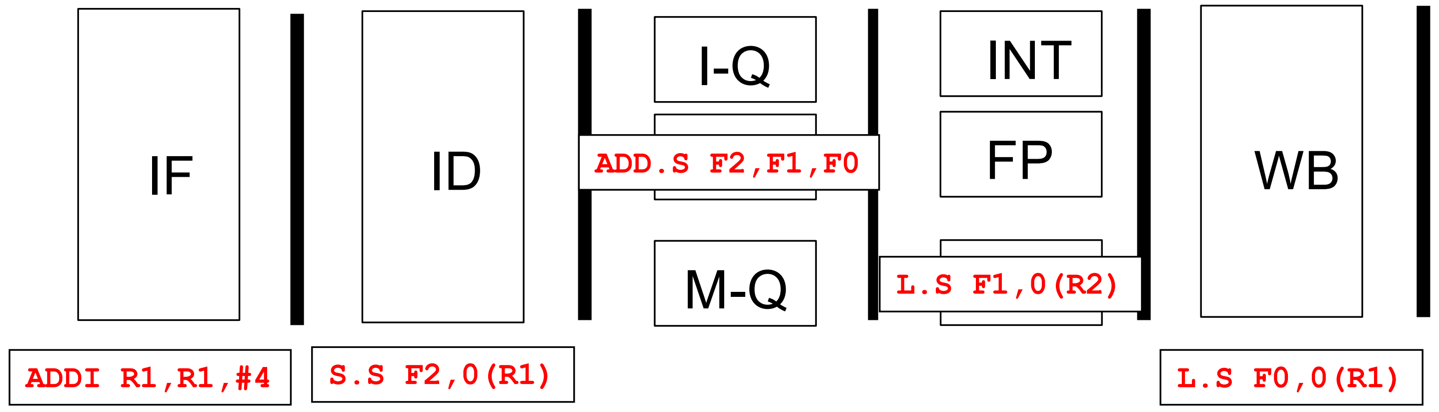


```
Loop     L.S F0,0(R1)
         L.S F1,0(R2)
         ADD.S F2,F1,F0
         S.S F2,0(R1)
         ADDI R1,R1,#4
         ADDI R2,R2,#4
         SUBI R3,R3,#1
         BNEZ R3,Loop
```

Cycle: 4



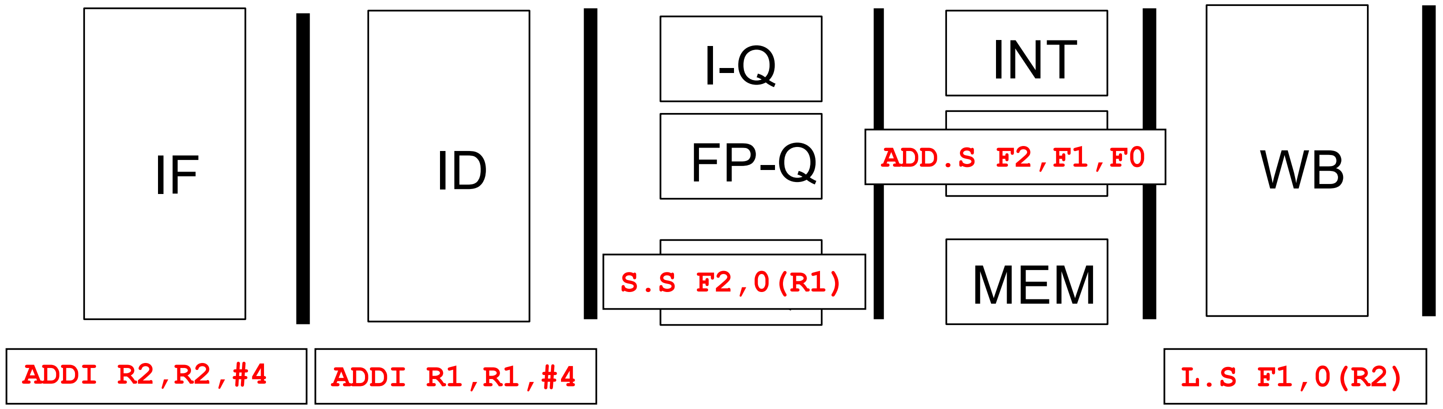
```
Loop      L.S F0,0(R1)
          L.S F1,0(R2)
          ADD.S F2,F1,F0
        ➡ S.S F2,0(R1)
          ADDI R1,R1,#4
          ADDI R2,R2,#4
          SUBI R3,R3,#1
          BNEZ R3,Loop
```



Cycle: 5

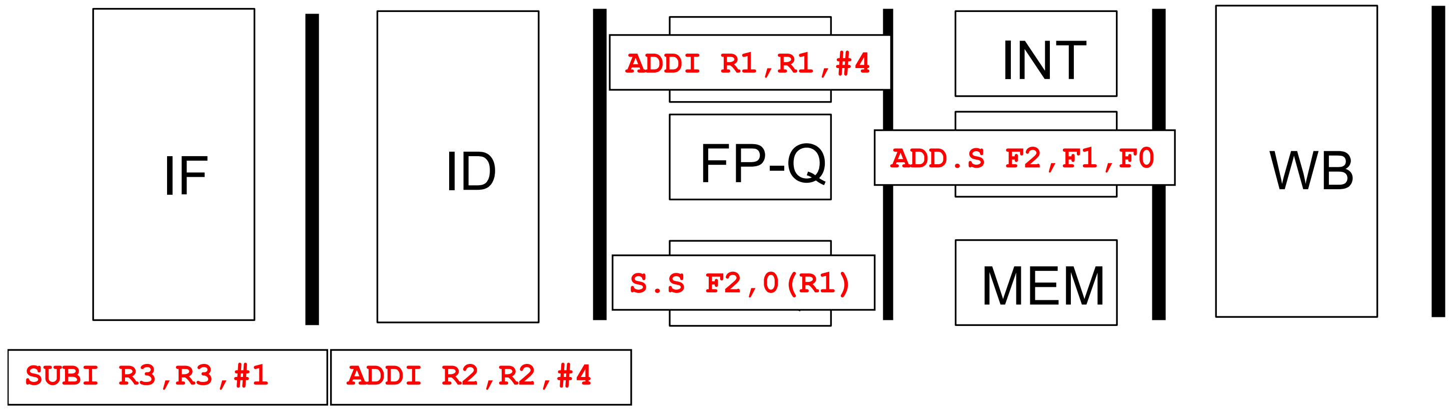
```
Loop    L.S F0,0(R1)
        L.S F1,0(R2)
        ADD.S F2,F1,F0
        S.S F2,0(R1)
      ➡ ADDI R1,R1,#4
        ADDI R2,R2,#4
        SUBI R3,R3,#1
        BNEZ R3,Loop
```

Cycle: 6

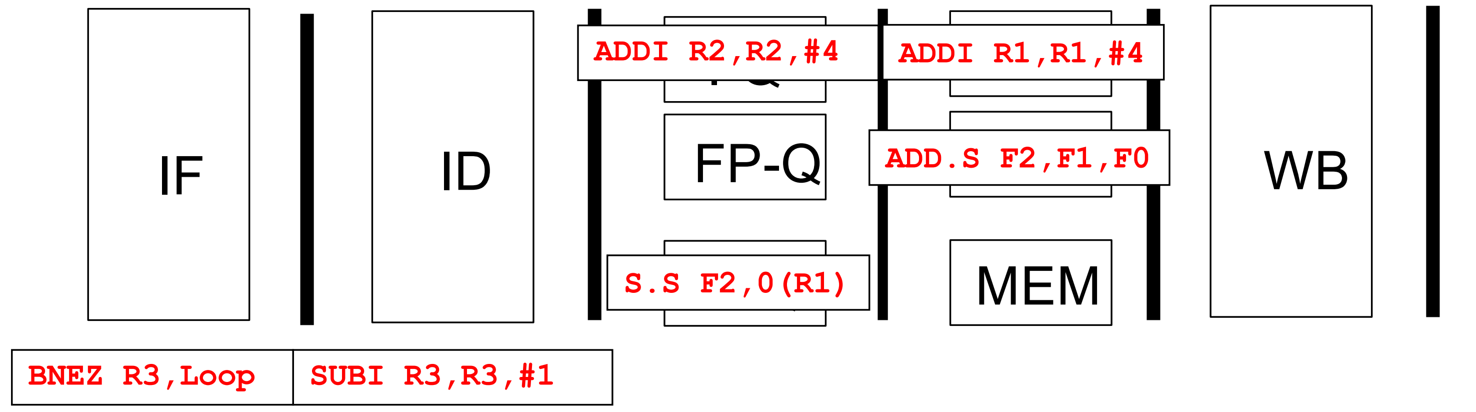


```
Loop    L.S F0,0(R1)
        L.S F1,0(R2)
        ADD.S F2,F1,F0
        S.S F2,0(R1)
        ADDI R1,R1,#4
      ➡ ADDI R2,R2,#4
        SUBI R3,R3,#1
        BNEZ R3,Loop
```

Cycle: 7



```
Loop    L.S F0,0(R1)
        L.S F1,0(R2)
        ADD.S F2,F1,F0
        S.S F2,0(R1)
        ADDI R1,R1,#4
        ADDI R2,R2,#4
      ➡ SUBI R3,R3,#1
        BNEZ R3,Loop
```

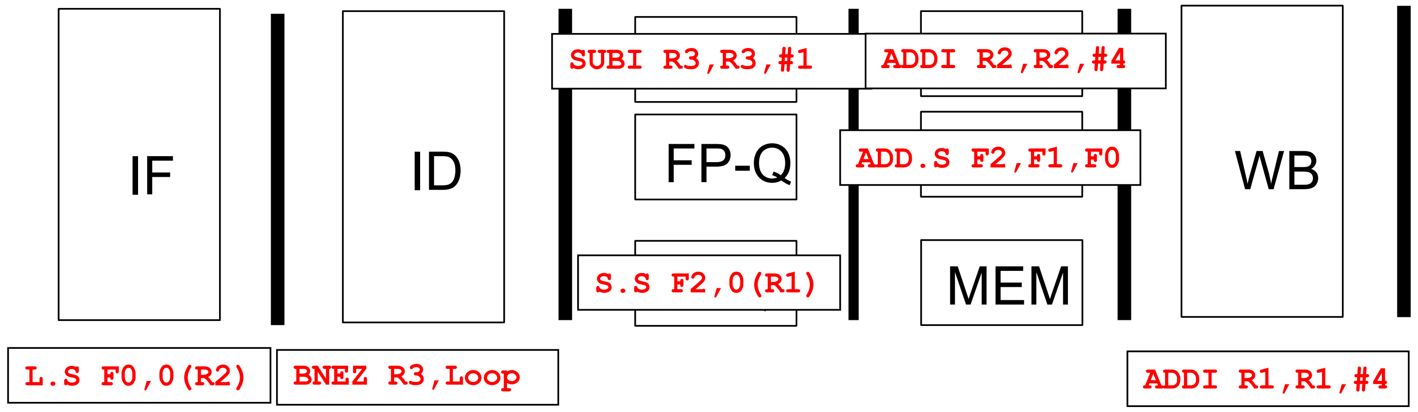


Cycle: 8

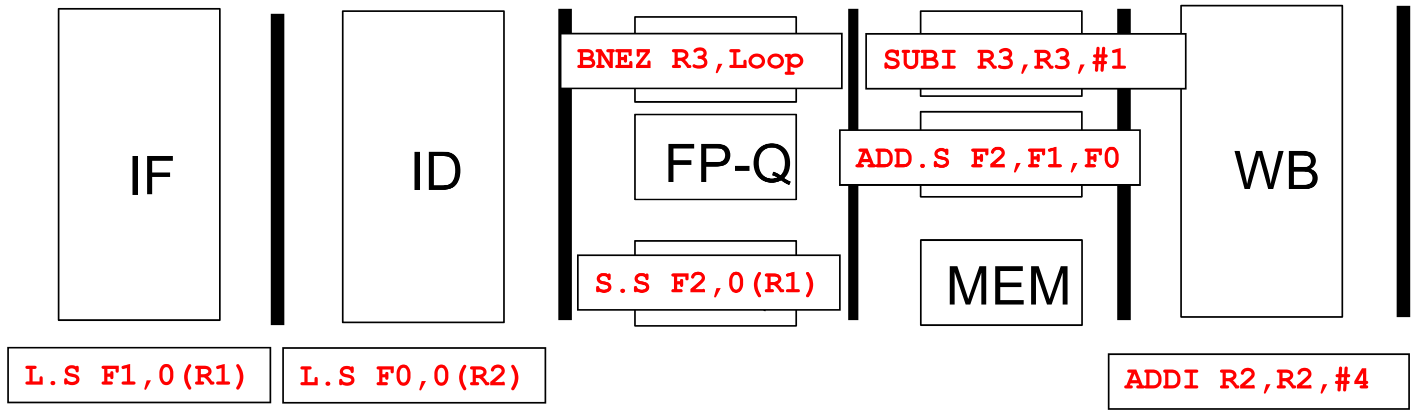
The ADDI R1,R1,#4 is being executed Out-of-program-order with respect to the S.S F2,0(R1)!

```
Loop    L.S F0,0(R1)
        L.S F1,0(R2)
        ADD.S F2,F1,F0
        S.S F2,0(R1)
        ADDI R1,R1,#4
        ADDI R2,R2,#4
        SUBI R3,R3,#1
      ➡ BNEZ R3,Loop
```

Cycle: 9



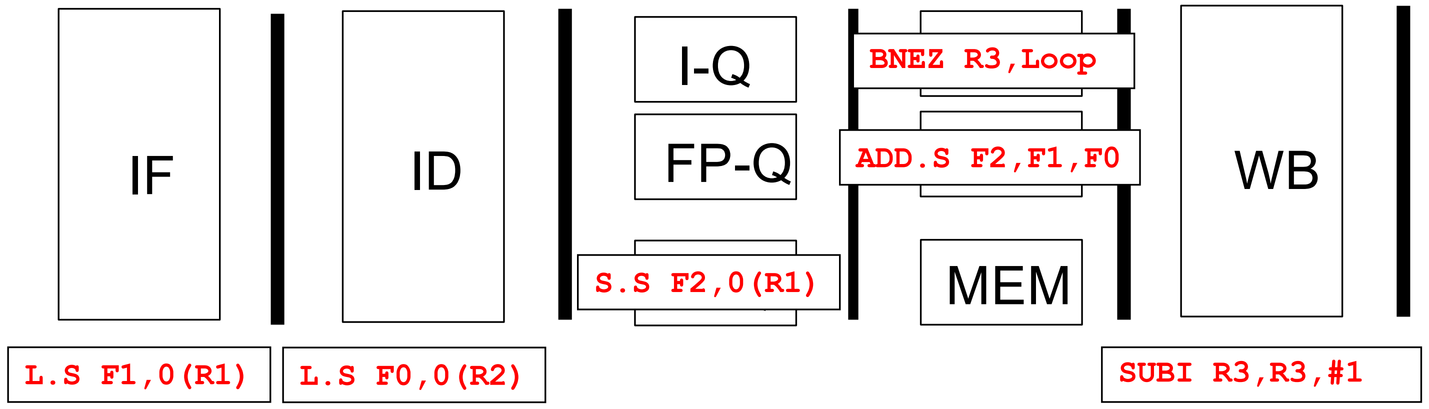
```
Loop ➡ L.S F0,0(R1)
       L.S F1,0(R2)
       ADD.S F2,F1,F0
       S.S F2,0(R1)
       ADDI R1,R1,#4
       ADDI R2,R2,#4
       SUBI R3,R3,#1
       BNEZ R3,Loop
```



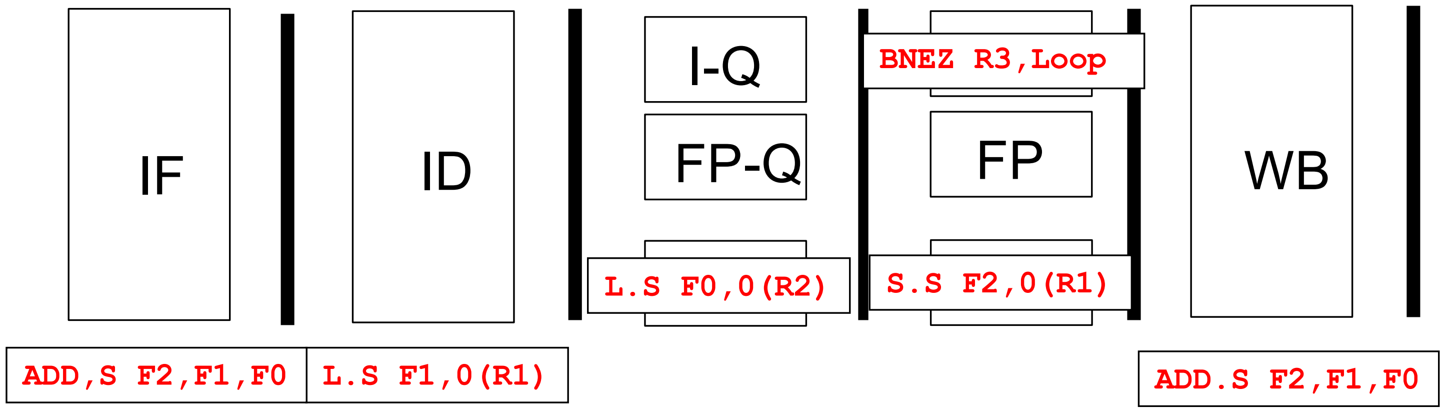
Cycle: 10

```
Loop     L.S F0,0(R1)
         L.S F1,0(R2)
         ADD.S F2,F1,F0
         S.S F2,0(R1)
         ADDI R1,R1,#4
         ADDI R2,R2,#4
         SUBI R3,R3,#1
         BNEZ R3,Loop
```

Cycle: 11



```
Loop      L.S F0,0(R1)
      ➡   L.S F1,0(R2)
          ADD.S F2,F1,F0
          S.S F2,0(R1)
          ADDI R1,R1,#4
          ADDI R2,R2,#4
          SUBI R3,R3,#1
          BNEZ R3,Loop
```



Cycle: 12

```
Loop    L.S F0,0(R1)
        L.S F1,0(R2)
      ➡ ADD.S F2,F1,F0
        S.S F2,0(R1)
        ADDI R1,R1,#4
        ADDI R2,R2,#4
        SUBI R3,R3,#1
        BNEZ R3,Loop
```

Question:
Consider the program below and determine how many cycles it takes to execute one iteration on the example pipeline.

```
Loop    L.S F0,0(R1)
        ADD.S F2,F1,F0
        S.S F2,0(R1)
        ADDI R1,R1,#4
        SUBI R3,R3,#1
        BNEZ R3,Loop
```

Answer:
We fill out the pipeline diagram below.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
I1	IF	ID	IS	EX	WB								
I2		IF	ID	IS	EX	EX	EX	EX	EX	WB			
I3			IF	ID	IS	IS	IS	IS	IS	EX	WB		
I4				IF	ID	IS	EX	WB					
I5					IF	ID	IS	EX	WB				
I6						IF	ID	IS	EX	WB	WB	WB	

```
Loop I1:L.S F0,0(R1)
     I2:ADD.S F2,F1,F0
     I3:S.S F2,0(R1)
     I4:ADDI R1,R1,#4
     I5:SUBI R3,R3,#1
     I6:BNEZ R3,Loop
```

Answer:
It takes 12 cycles

Question:
In what order are the instructions fetched and in what order are they completed

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
I1	IF	ID	IS	EX	WB								
I2		IF	ID	IS	EX	EX	EX	EX	EX	WB			
I3			IF	ID	IS	IS	IS	IS	IS	EX	WB		
I4				IF	ID	IS	EX	WB					
I5					IF	ID	IS	EX	WB				
I6						IF	ID	IS	EX	WB	WB	WB	

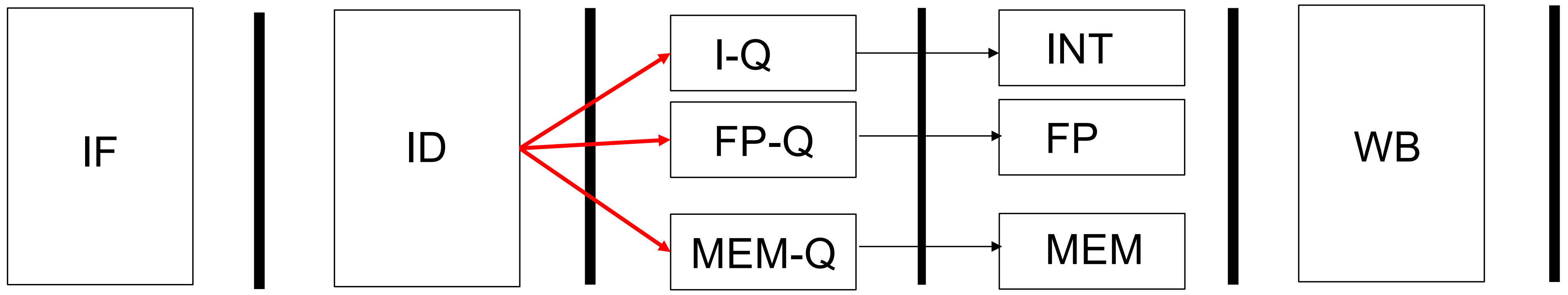
Answer:
Fetch order = program order: I1,I2,I3,I4,I5
Completion order: I1, I4,I5,I2,I3,I6

Tomasulo algorithm

Structural hazards

Data hazards

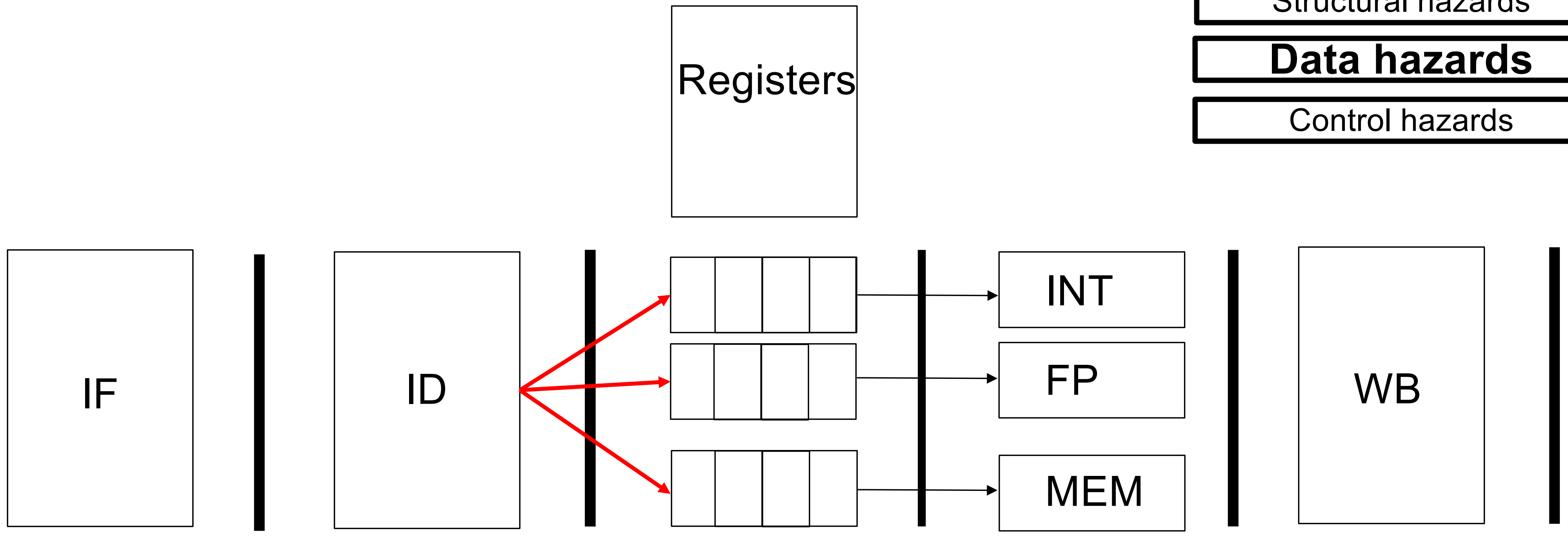
Control hazards



Structural hazards

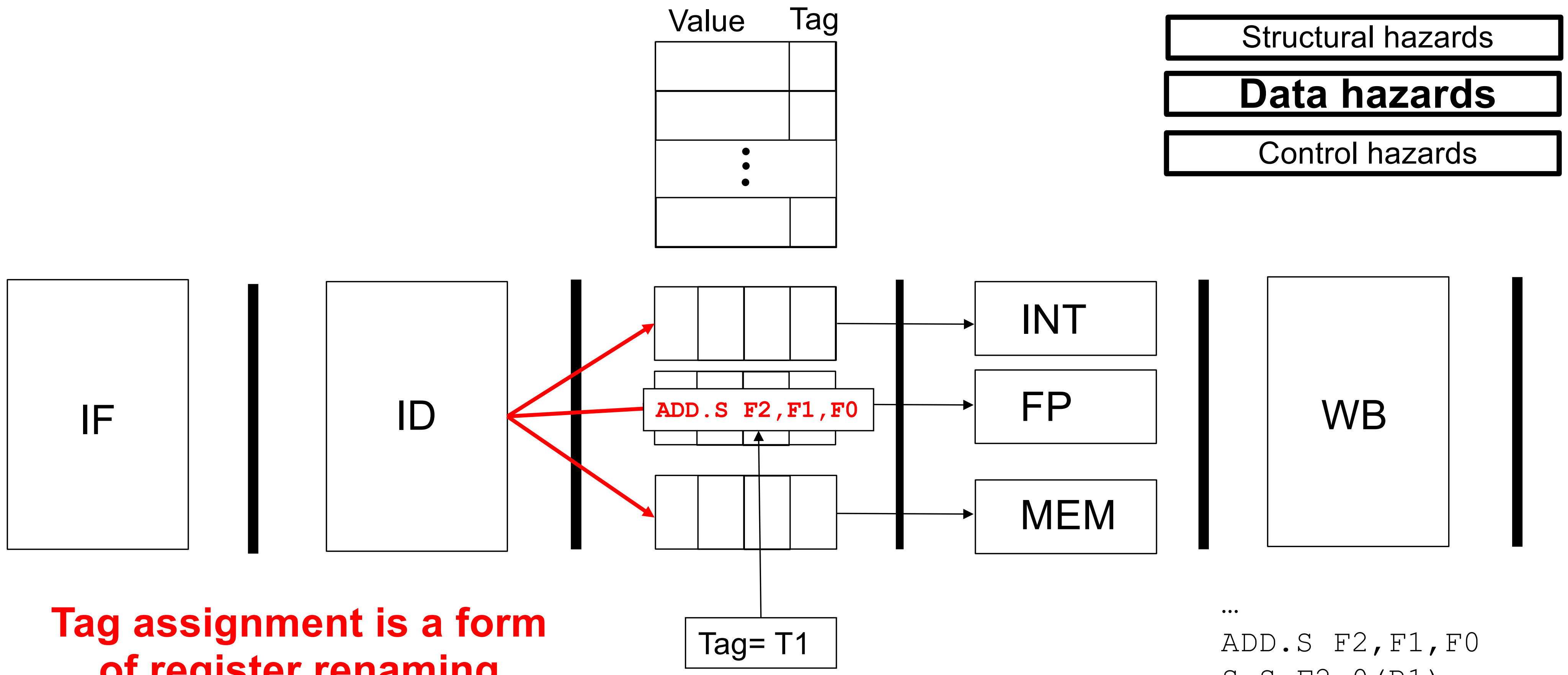
Data hazards

Control hazards



Out-of-order execution => All data hazards show up!

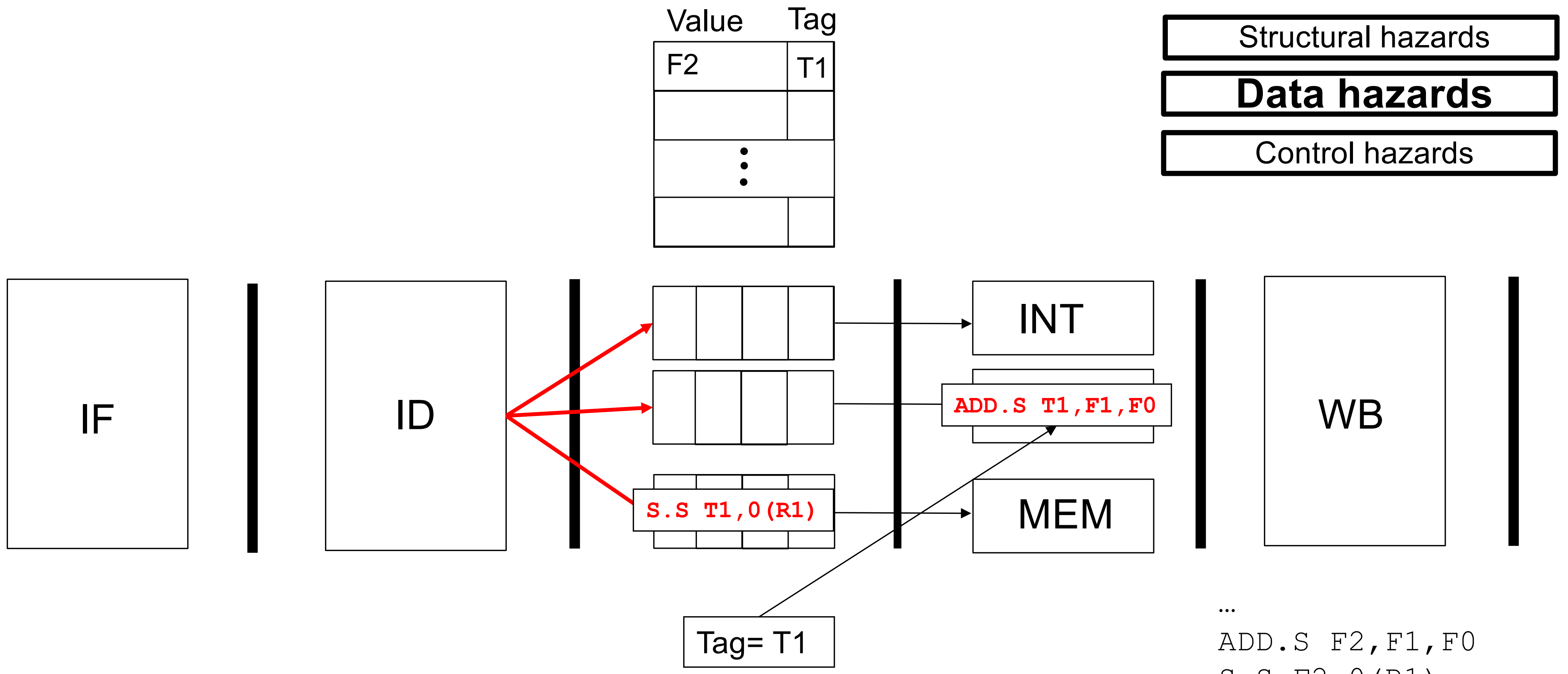
Key recipe to resolve data hazards: register renaming!



Tag assignment is a form of register renaming

```
…
ADD.S  F2,F1,F0
S.S  F2,0(R1)
…
```

The Tomasulo Algorithm



Structural hazards

Data hazards

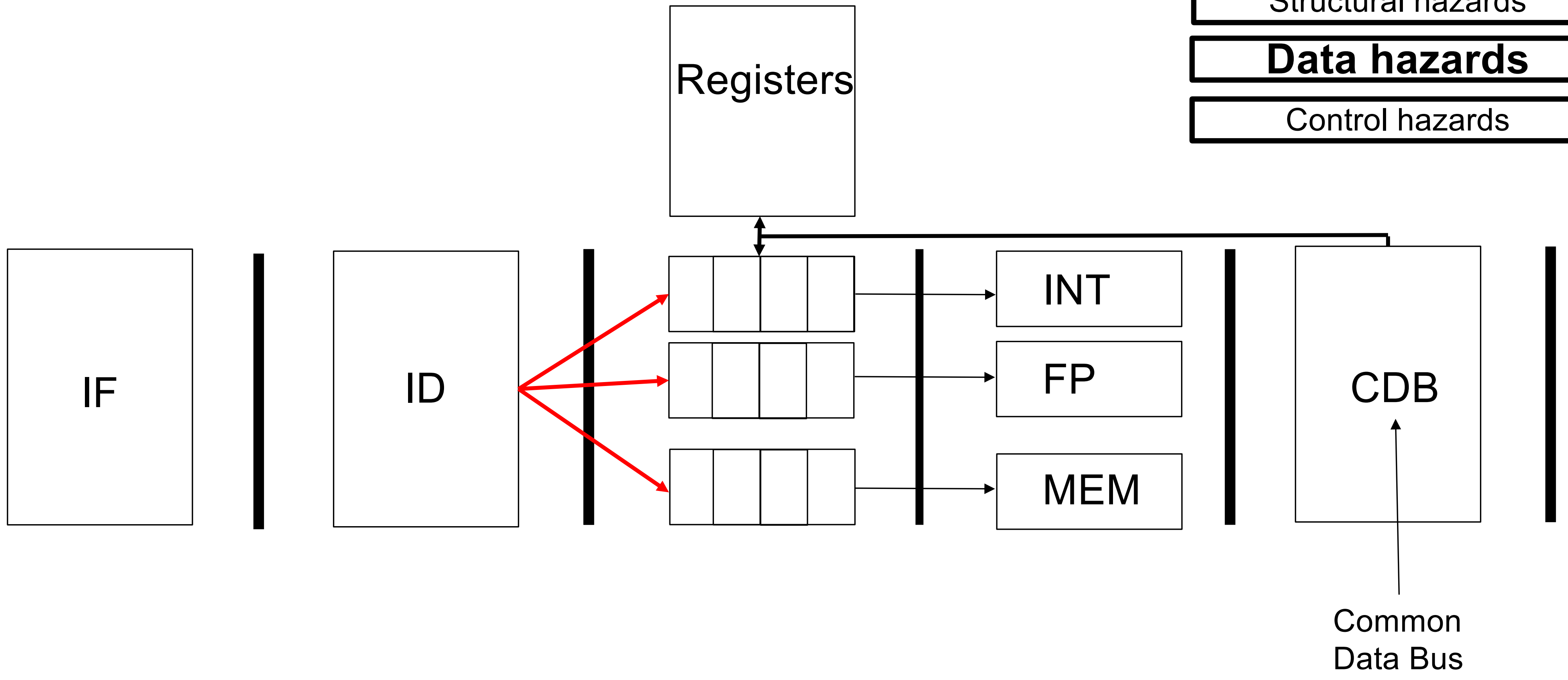
Control hazards

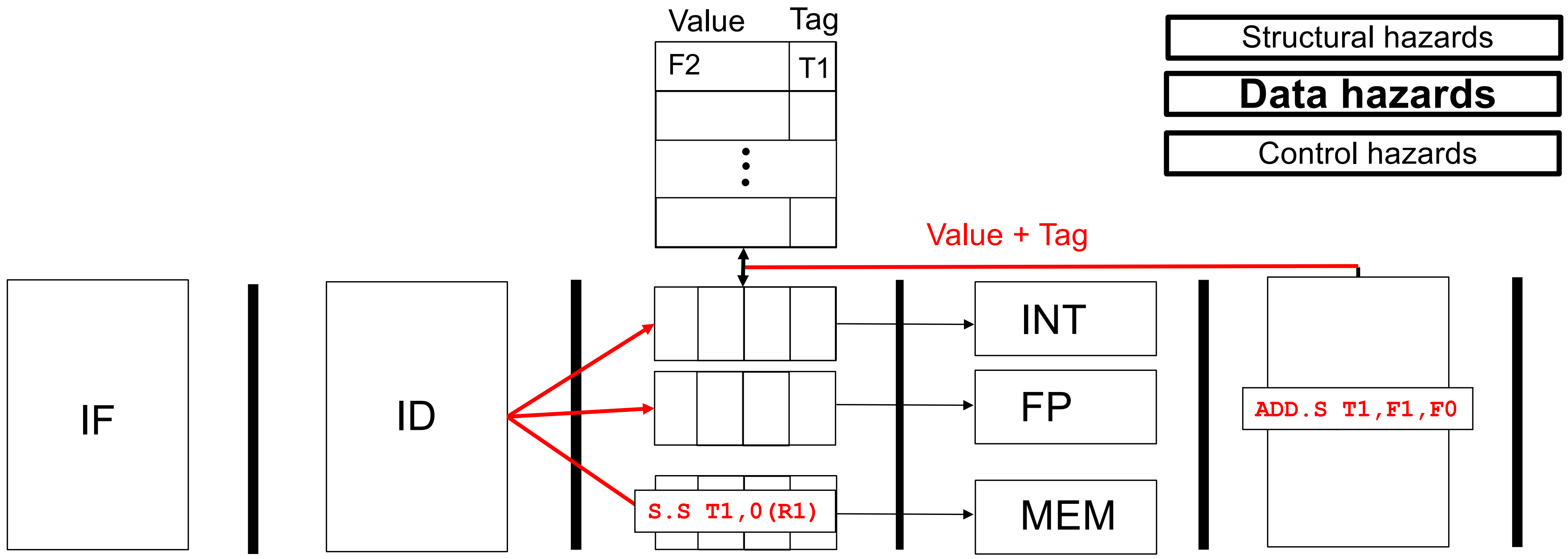
```
…
ADD.S  F2,F1,F0
S.S  F2,0(R1)
…
```

Structural hazards

Data hazards

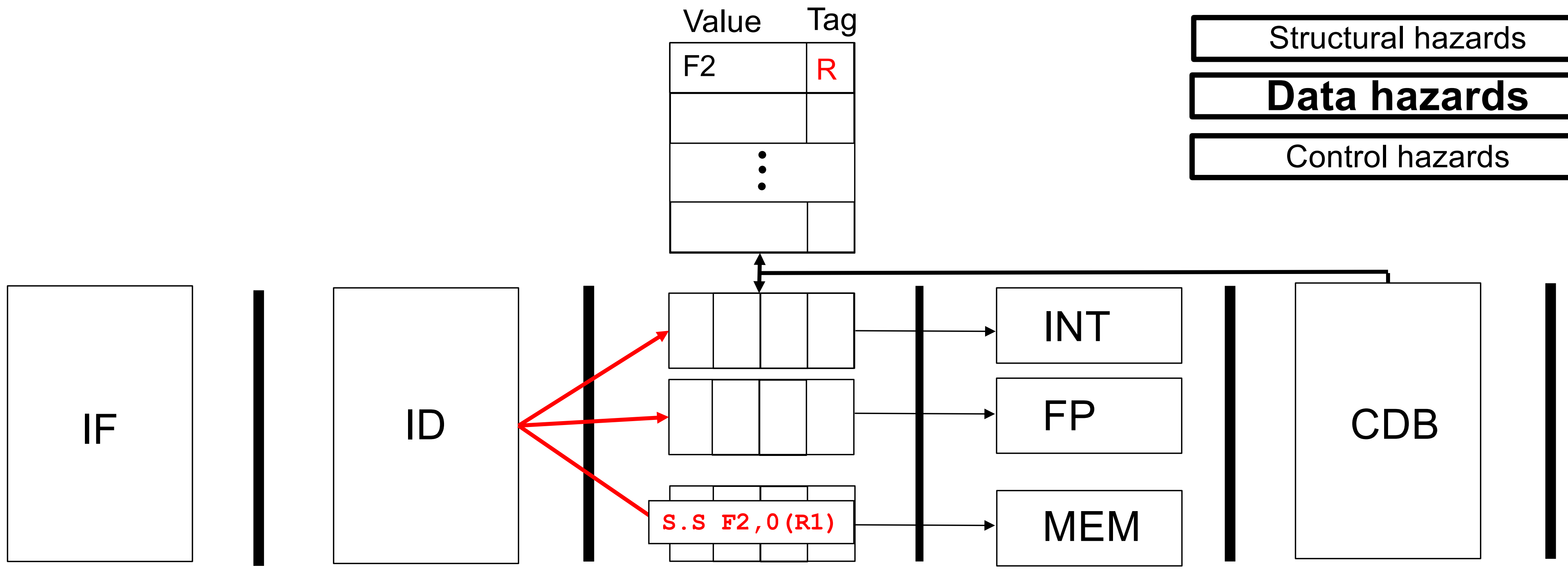
Control hazards





Register renaming

```
…
ADD.S F2,F1,F0
S.S F2,0(R1)
…
```

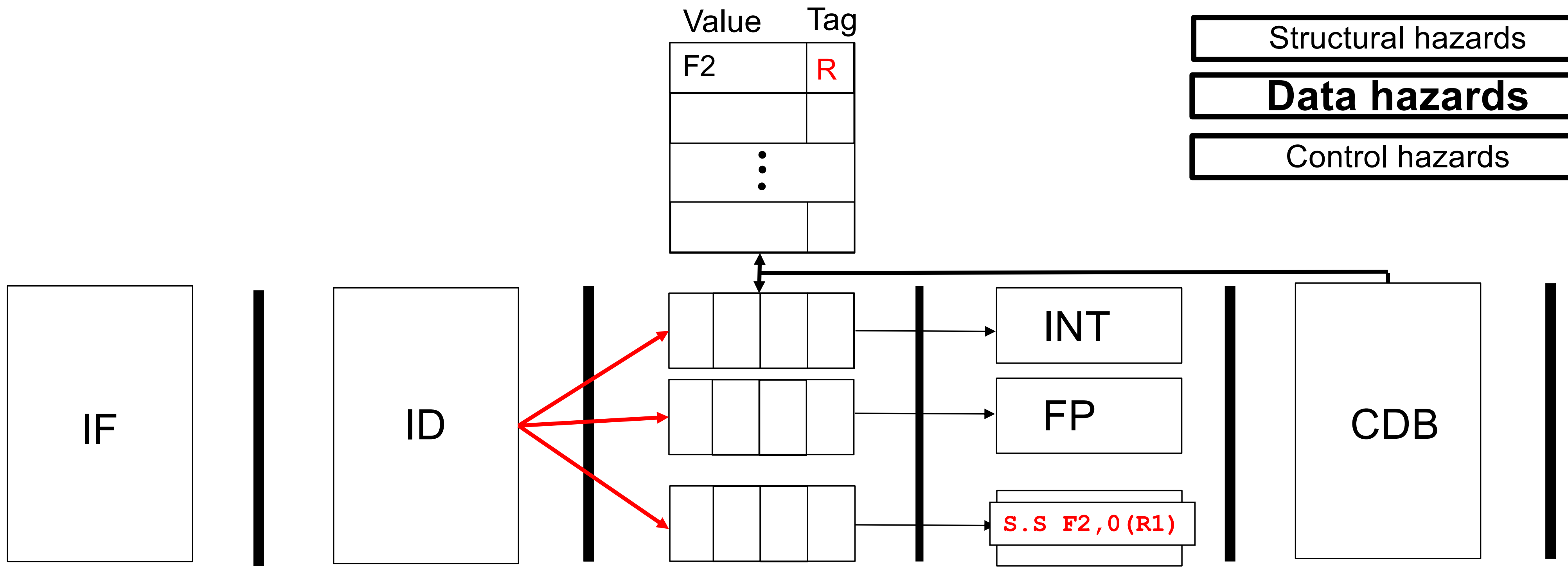


Structural hazards

Data hazards

Control hazards

```
…
ADD.S  F2,F1,F0
S.S  F2,0(R1)
…
```



Structural hazards

Data hazards

Control hazards

```
…
ADD.S  F2,F1,F0
S.S  F2,0(R1)
…
```

Resolution of Data Hazards in the Tomasulo algorithm

Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)

```
ADD.S F2,F1,F0
S.S F2, 0(R1)
```

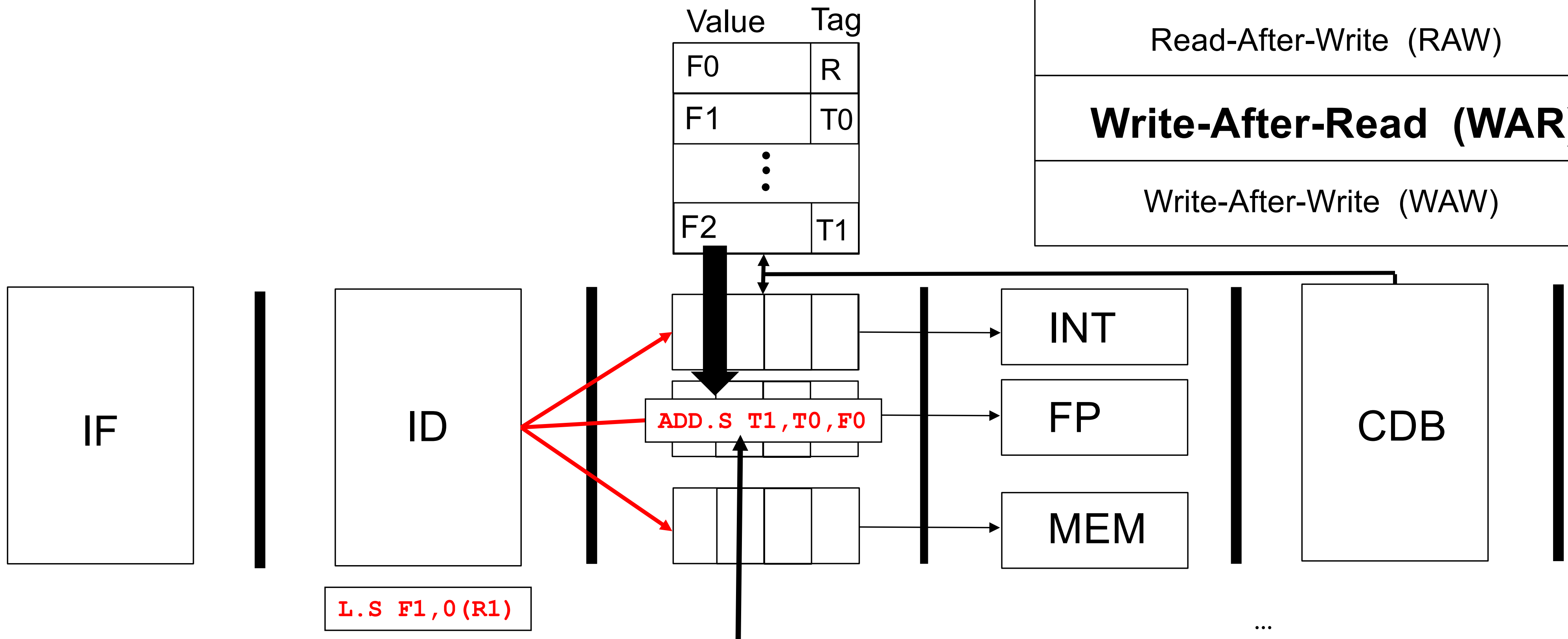
Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)

```
ADD.S F1,F3,F4
ADD.S F2,F1,F0
L.S F1,0(R1)
```

Read-After-Write (RAW)

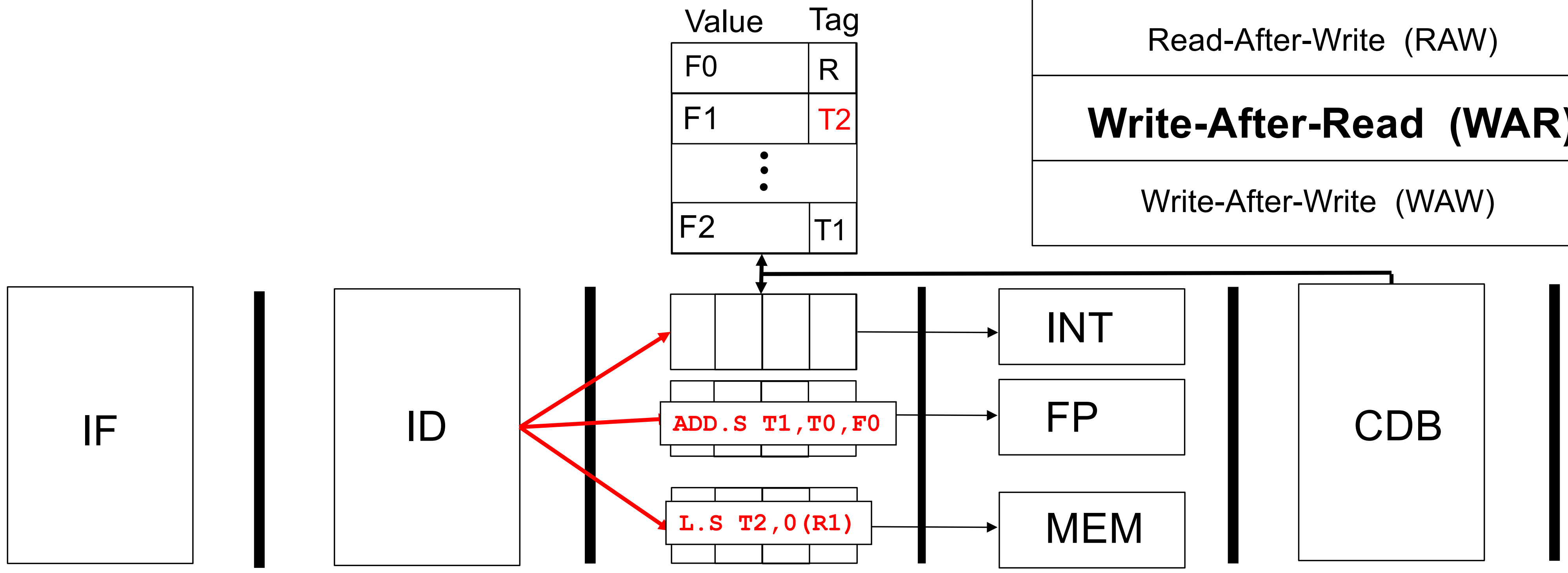
Write-After-Read (WAR)

Write-After-Write (WAW)



This ADD waits for a previous instruction to provide F1 through tag T0

```
…
ADD.S F2,F1,F0
L.S F1,0(R1)
…
```

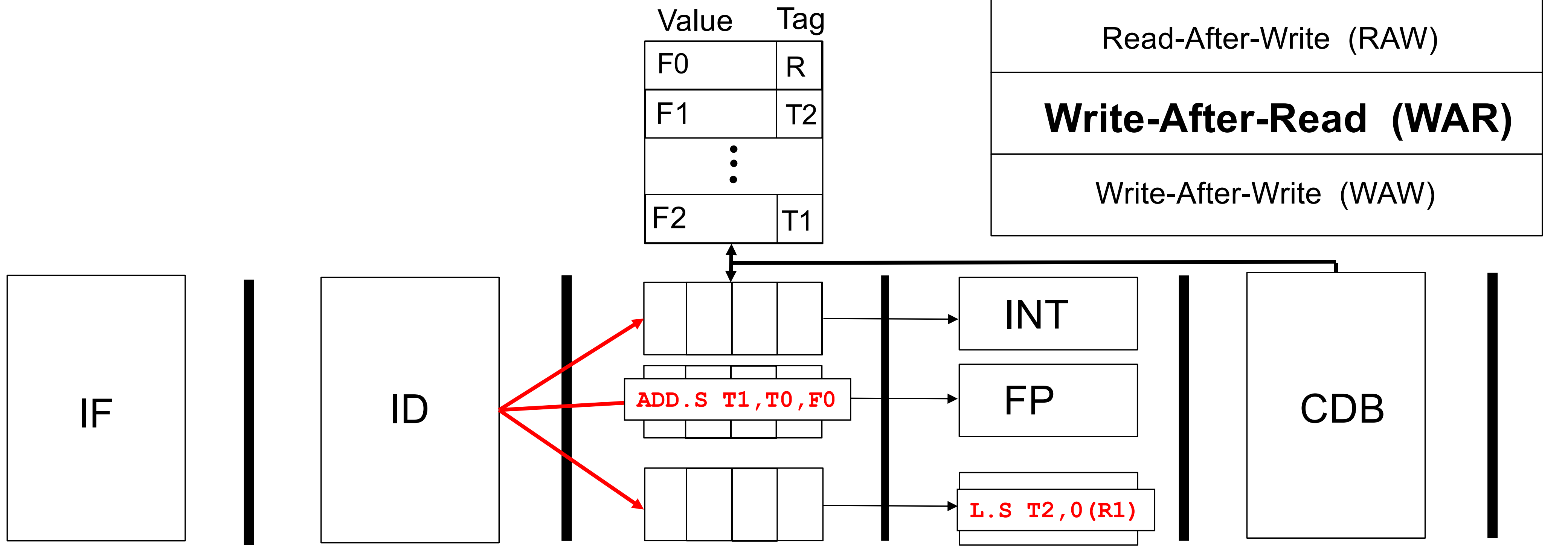


Read-After-Write (RAW)

Write-After-Read (WAR)

Write-After-Write (WAW)

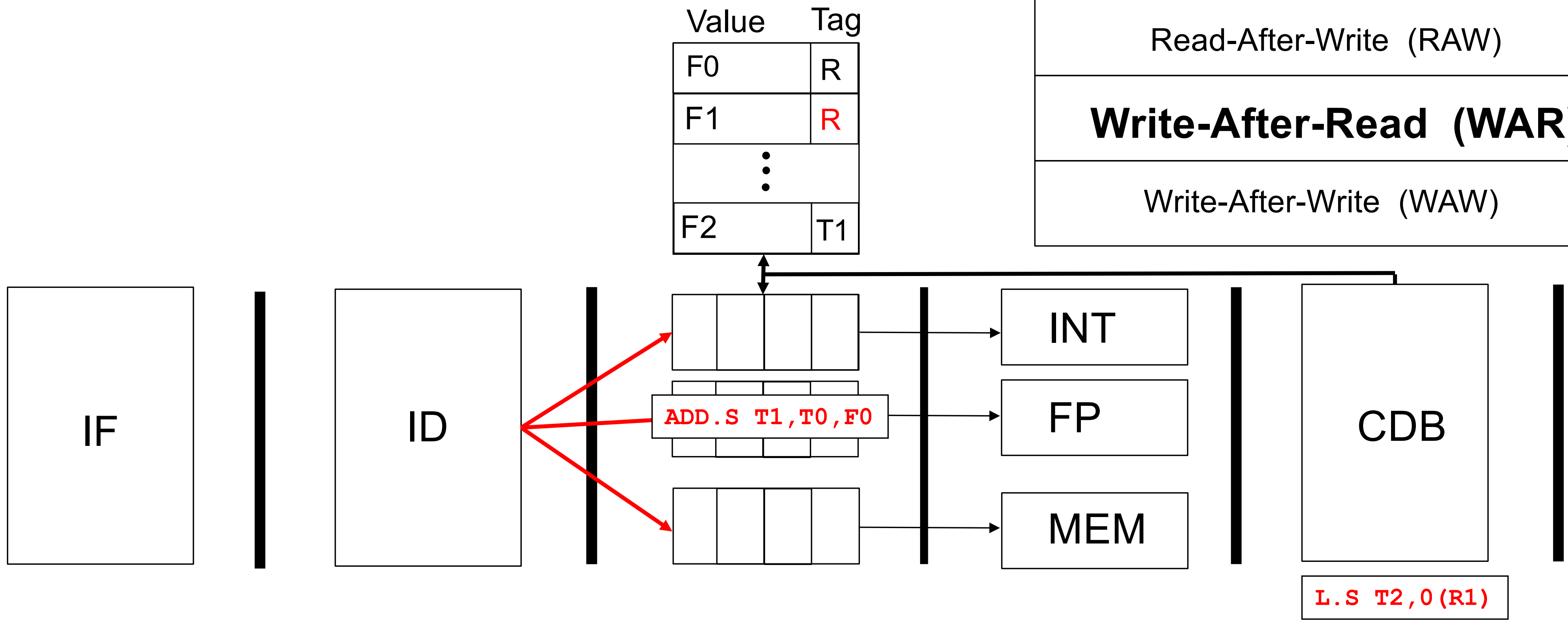
```
…
ADD.S F2,F1,F0
L.S F1,0(R1)
…
```



Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)

```
…
ADD.S F2,F1,F0
L.S F1,0(R1)
…
```

Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)



```
…
ADD.S  F2,F1,F0
L.S  F1,0(R1)
…
```

Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)

```
ADD.S F2,F1,F0
L.S F2,0(R1)
```

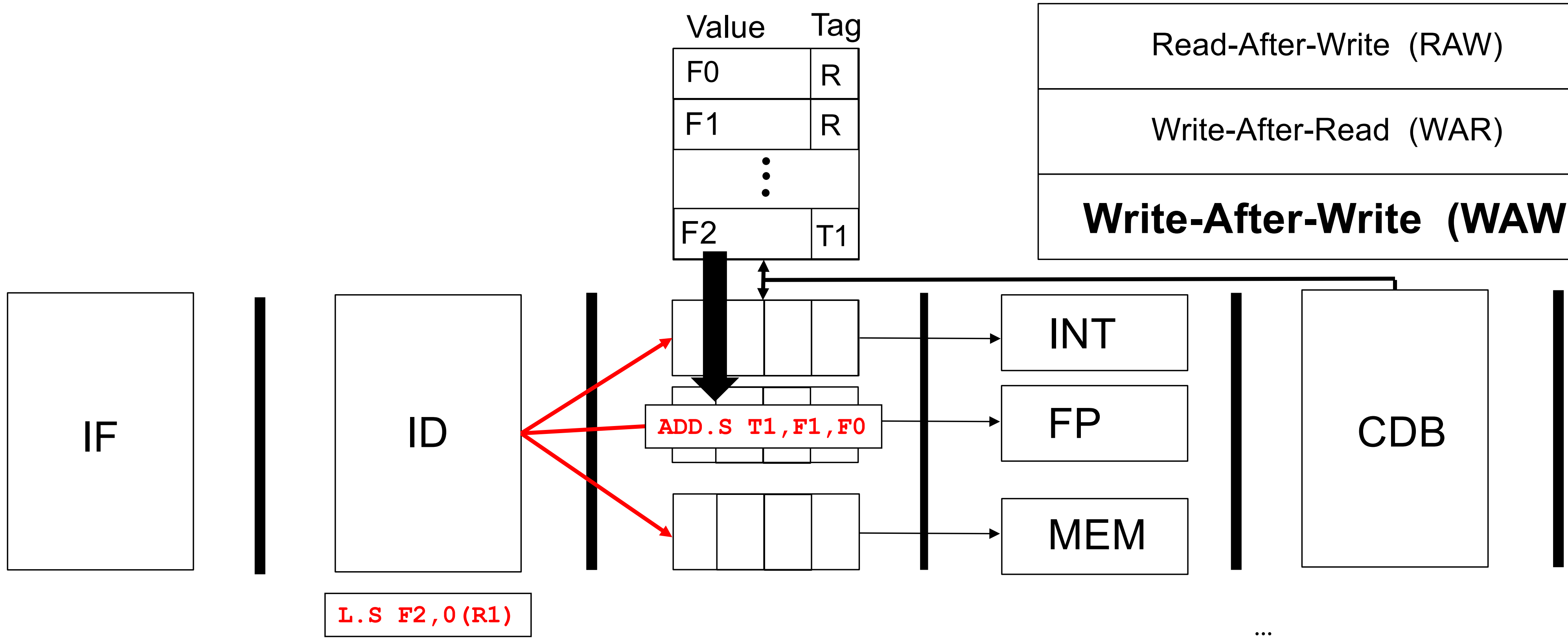
Question:
How is this WAW hazard eliminated by Tomasulo algorithm?

Answer:
We will demonstrate this next.

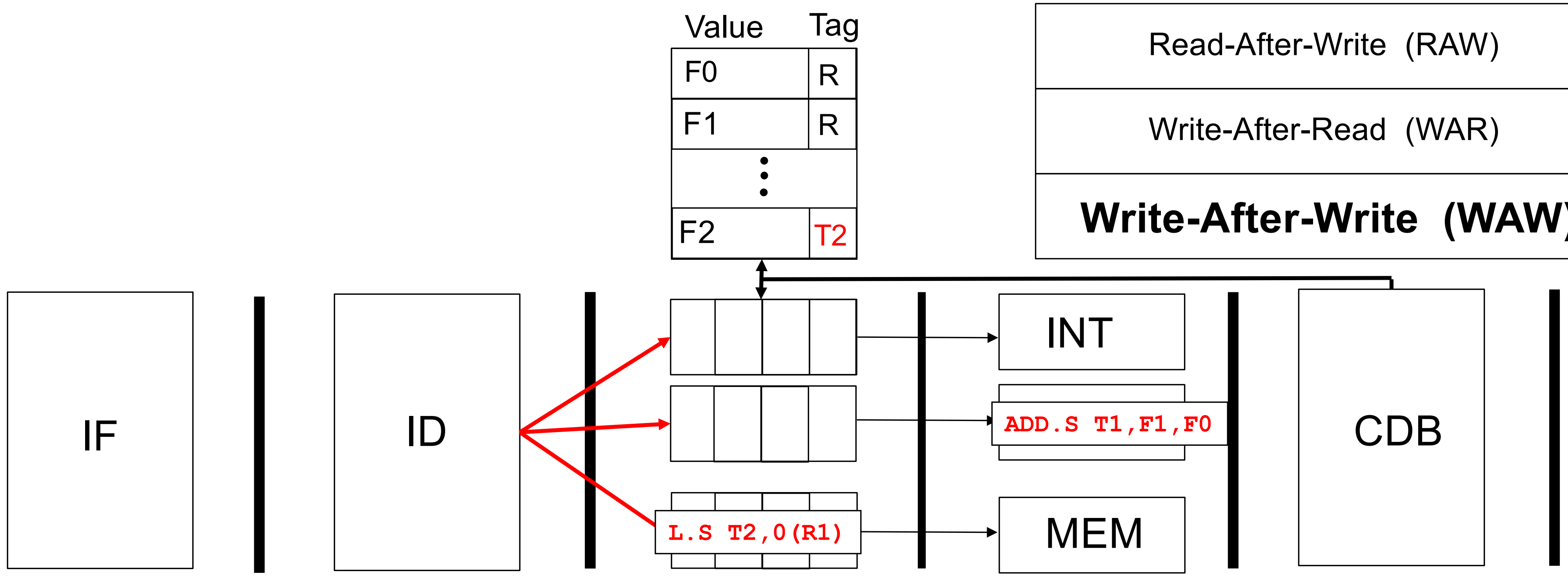
Read-After-Write (RAW)

Write-After-Read (WAR)

Write-After-Write (WAW)

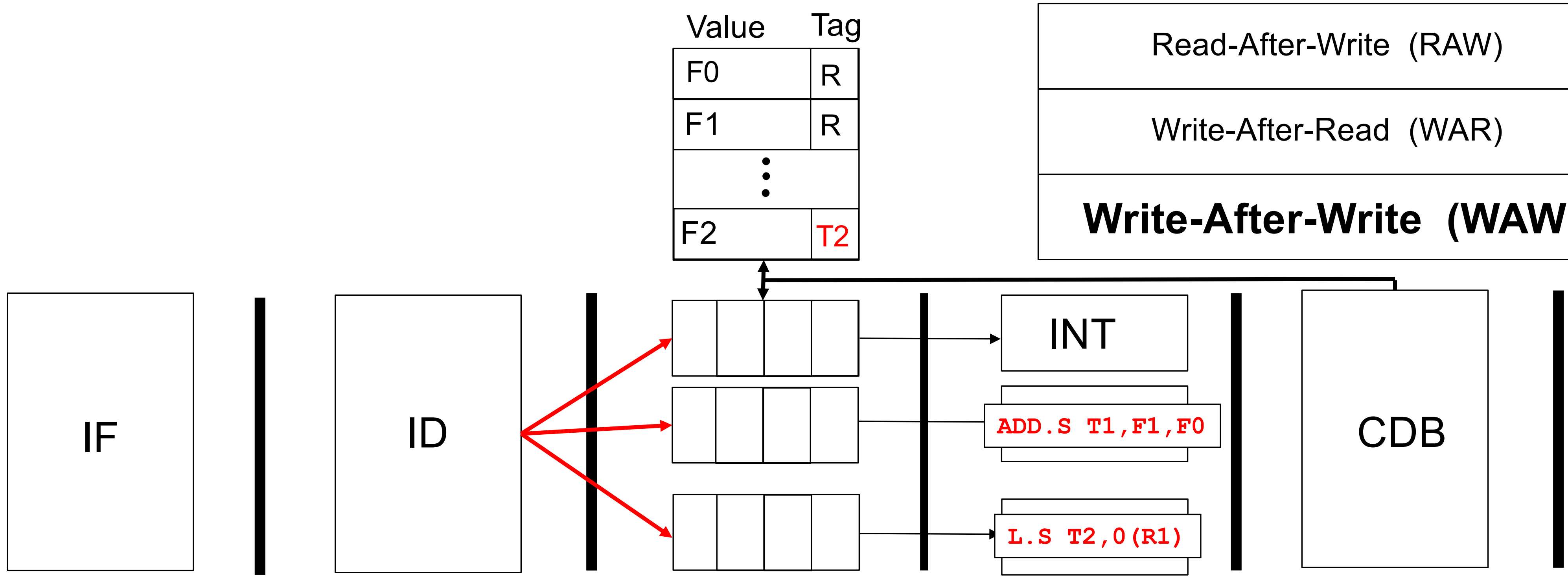


```
…
ADD.S F2,F1,F0
L.S F2,0(R1)
…
```



Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)

```
…
ADD.S F2,F1,F0
L.S F2,0(R1)
…
```



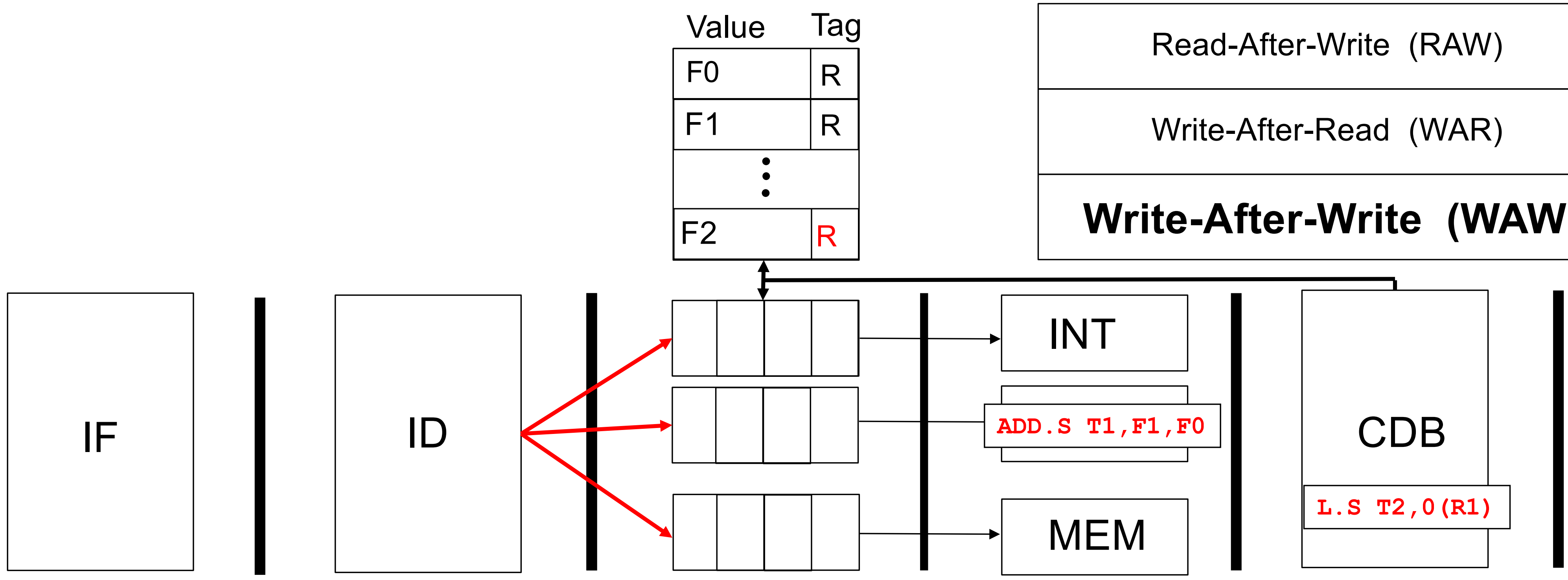
Read-After-Write (RAW)
Write-After-Read (WAR)
Write-After-Write (WAW)

```
…
ADD.S F2,F1,F0
L.S F2,0(R1)
…
```

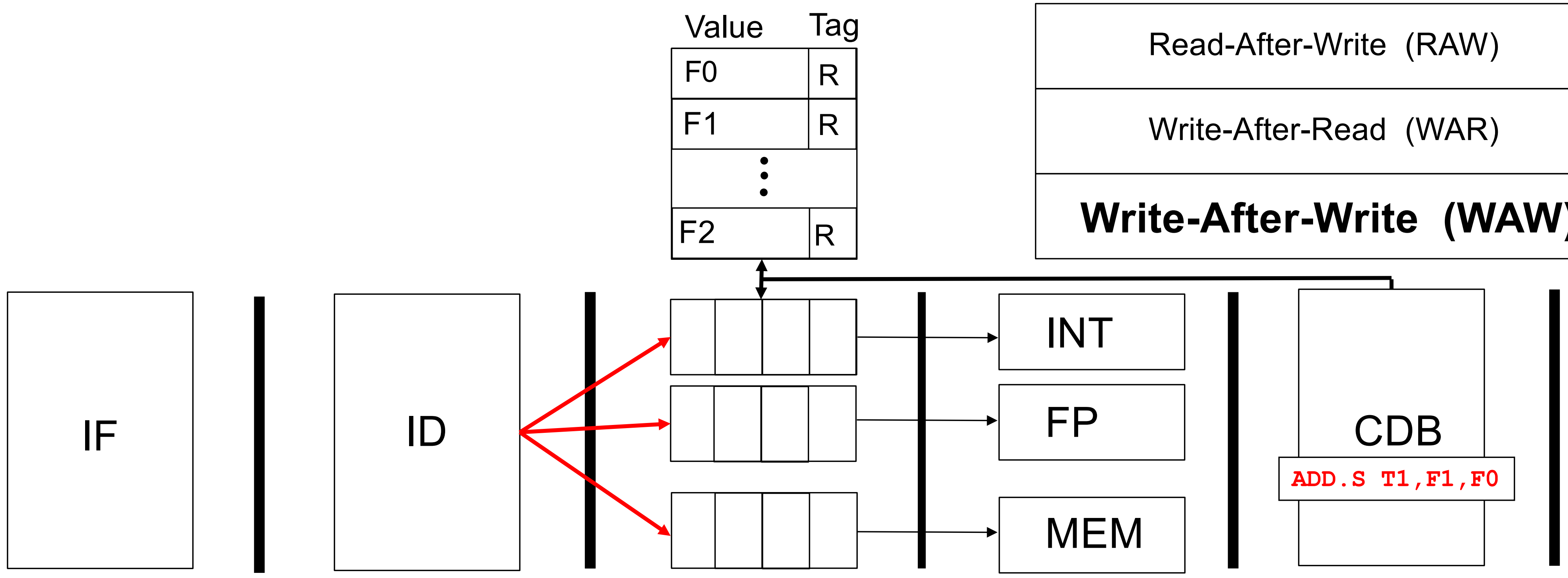
Read-After-Write (RAW)

Write-After-Read (WAR)

Write-After-Write (WAW)



```
…
ADD.S F2,F1,F0
L.S F2,0(R1)
…
```



Read-After-Write (RAW)

Write-After-Read (WAR)

Write-After-Write (WAW)

```
…
ADD.S  F2,F1,F0
L.S  F2,0(R1)
…
```

Question:
Consider the following code segment:
I1: ADD F1,F2,F3
I2: ADD F2,F4,F5
How many cycles will it take until I1 and I2 are completed assuming there are two floating-point execution units?

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
I1	IF	ID	IS	EX	EX	EX	EX	EX	WB				
I2		IF	ID	IS	EX	EX	EX	EX	EX	WB			

Answer:
In 9 and 10 cycles, respectively