Exercise Session 3

Dynamic Branch Prediction, Complex Pipelining, Pipelining backup

Advanced Computer Architectures

Politecnico di Milano

March 19th, 2024

Alessandro Verosimile <alessandro.verosimile@polimi.it>

POLITECNICO MILANO 1863

POLITECNICO MILANO 1863 NECST laboratory

Recall: Pipeline performance

Pipeline CPI = Ideal pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls

- **Ideal pipeline CPI**: measure of the maximum performance attainable by the implementation
- **Structural hazards**: HW cannot support this combination of instructions
- **Data hazards**: Instruction depends on result of prior instruction still in the pipeline
- **Control hazards**: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches, jumps, exceptions)

Recall: Three Classes of Hazards

Structural Hazards: Attempt to use the same resource from different instructions simultaneously

Example: Single memory for instructions and data

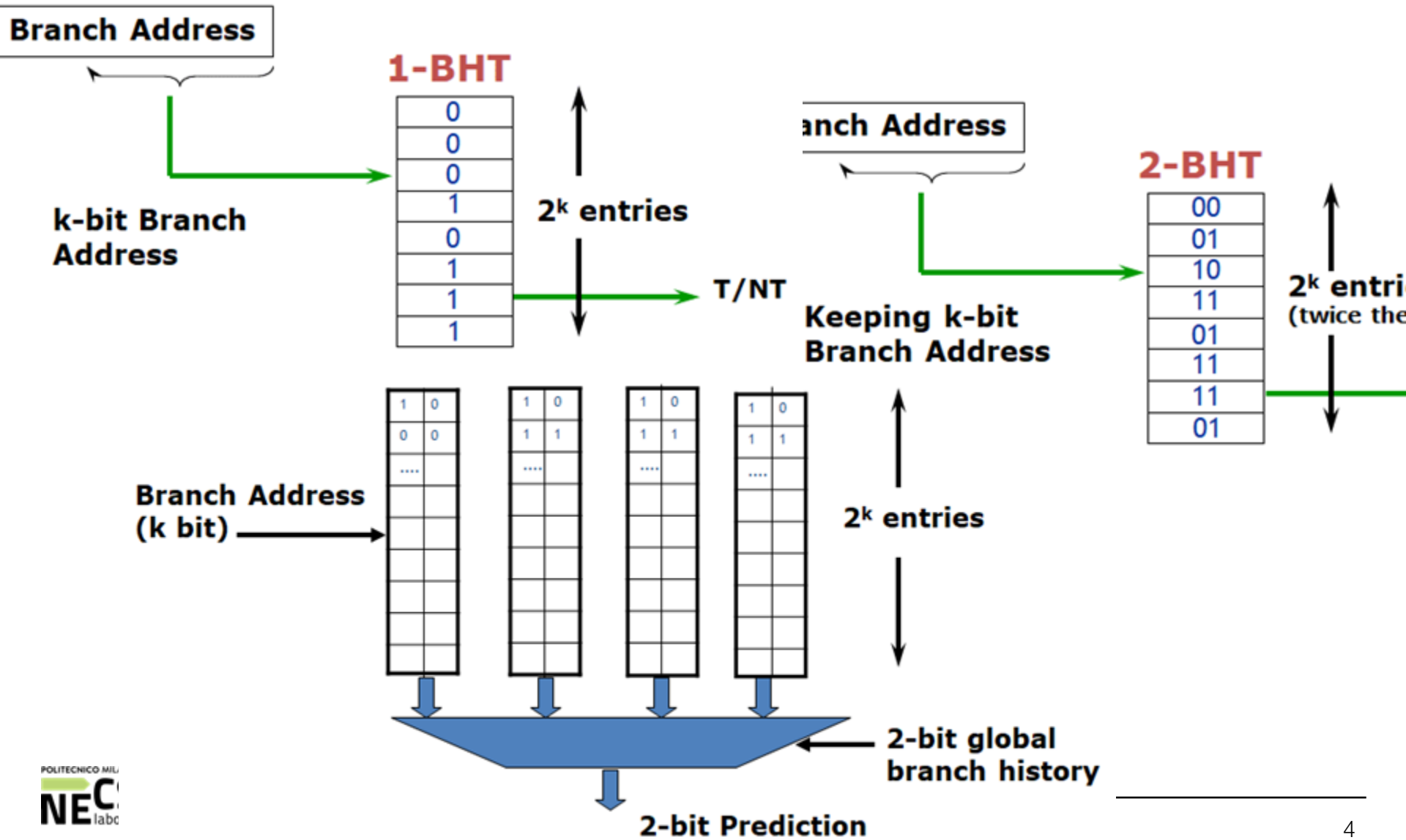
Data Hazards: Attempt to use a result before it is ready

Example: Instruction depending on a result of a previous instruction still in the pipeline

Control Hazards: Attempt to make a decision on the next instruction to execute before the condition is evaluated

Example: Conditional branch execution

Recall: Dynamic Branch Prediction



Dynamic Branch Predictor

- Describe (the answer has to be effectively supported) a 1-BHT and a 2-BHT able to execute the following assembly code (R0 is set to 2000, R1 is set to 0)

```
LOOP:     LD     F1     0      R0
          ADDD   F2     F1     F1
          ADDI   R1     R1     100
LOOP2:    MULTD  F2     F2     F1
          SUBI   R1     R1     1
          BNEZ   R1     LOOP2
          SUBI   R0     R0     2
          BNEZ   R0     LOOP
```

- The obtained result, in terms of mispredictions, is inline with theoretical characteristics of the two predictors? Please effectively support your answer.

A First Consideration

```
LOOP:    LD      F1      0       R0
         ADDD    F2      F1      F1
         ADDI    R1      R1      100
LOOP2:   MULTD   F2      F2      F1
         SUBI    R1      R1      1
         BNEZ    R1 LOOP2
         SUBI    R0      R0      2
         BNEZ    R0 LOOP
```

How many iterations?

R0 is set to 2000
R1 is set to 0

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1 LOOP2
        SUBI    R0      R0      2
        BNEZ    R0 LOOP
```

How many iterations?

R0 is set to 2000
R1 is set to 0

```
LOOP:   LD      F1   0    R0
        ADDD    F2   F1   F1
        ADDI    R1   R1   100
LOOP2:  MULTD   F2   F2   F1
        SUBI    R1   R1   1
        BNEZ    R1 LOOP2
        SUBI    R0   R0   2
        BNEZ    R0 LOOP
```

LOOP2
@T0 100 iterations

How many iterations?

R0 is set to 2000
R1 is set to 0

```
LOOP:   LD      F1   0    R0
        ADDD    F2   F1   F1
        ADDI    R1   R1   100
LOOP2:  MULTD   F2   F2   F1
        SUBI    R1   R1   1
        BNEZ    R1 LOOP2
        SUBI    R0   R0   2
        BNEZ    R0 LOOP
```

LOOP2
@T0 100 iterations

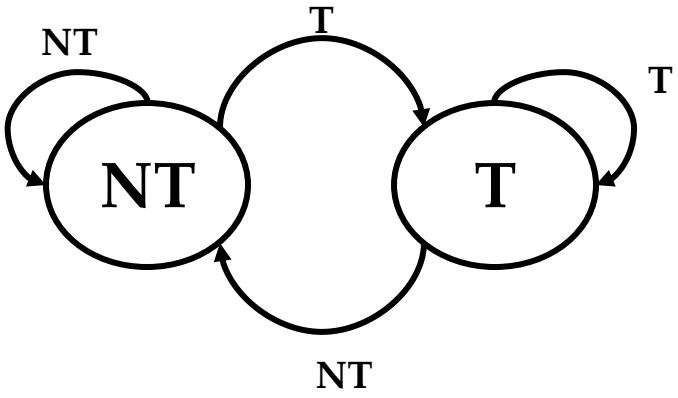
LOOP
1000 iterations

1bit - BHT

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000

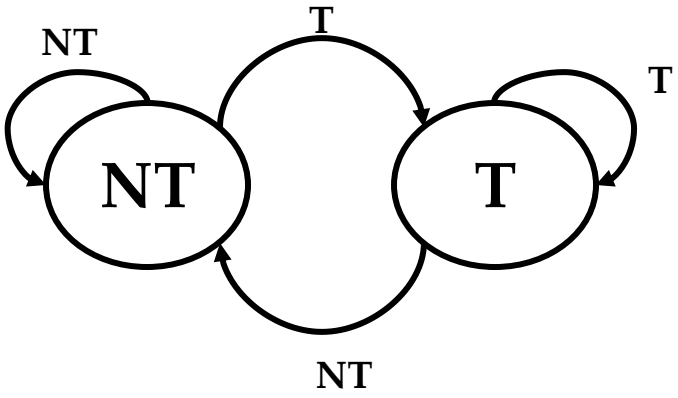
R1 is set to 0

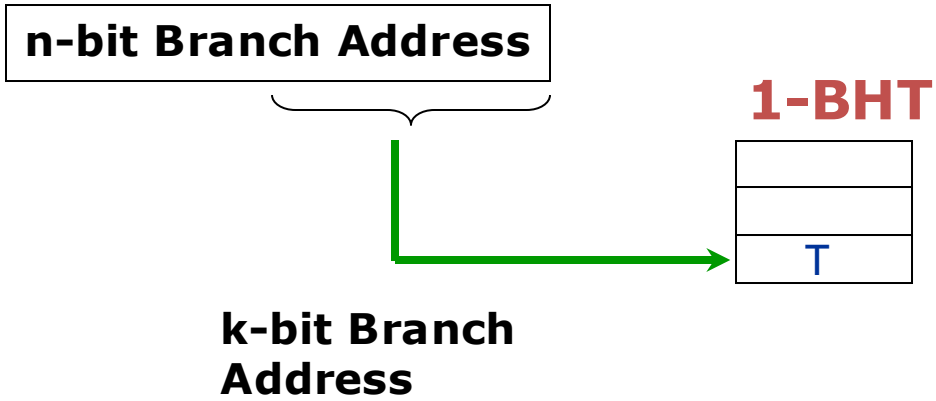


1bit - BHT

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000
R1 is set to 0





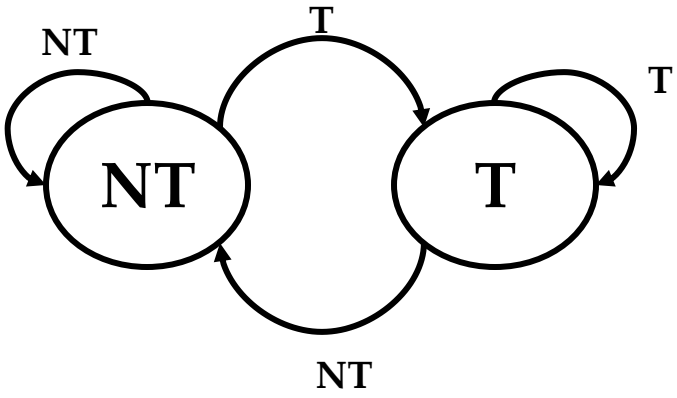
k-bit Branch Address:
Collide
Not collide

1bit - BHT - Not Collide

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000

R1 is set to 0



Let us consider that the branch addresses do not collide

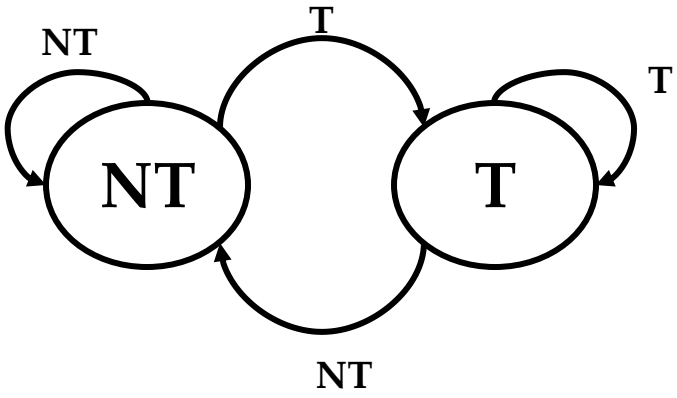
	1-BHT	1-BHT	1-BHT	1-BHT
LOOP:	T	T	NT	NT
LOOP2:	T	NT	T	NT

1bit - BHT Misprediction

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000

R1 is set to 0



Let us consider that the branch addresses do not collide

LOOP2
100 iterations

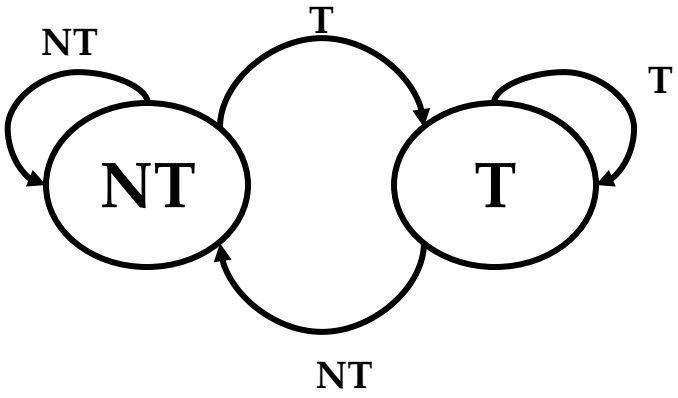
LOOP
1000 iterations

	1-BHT	1-BHT	1-BHT	1-BHT
LOOP:	T	T	NT	NT
LOOP2:	T	NT	T	NT
	1 + 1 + (1000-1) * 2	1 + 1000 * 2	2 + 1 + (1000-1) * 2	2 + 1000*2

1bit - BHT - Collision

LOOP:	LD	F1	0	R0
	ADDD	F2	F1	F1
	ADDI	R1	R1	100
LOOP2:	MULTD	F2	F2	F1
	SUBI	R1	R1	1
	BNEZ	R1	LOOP2	
	SUBI	R0	R0	2
	BNEZ	R0	LOOP	

R0 is set to 2000
R1 is set to 0



Let us consider that the branch addresses do collide

LOOP2
100 iterations

LOOP
1000 iterations

1-BHT

T

1-BHT

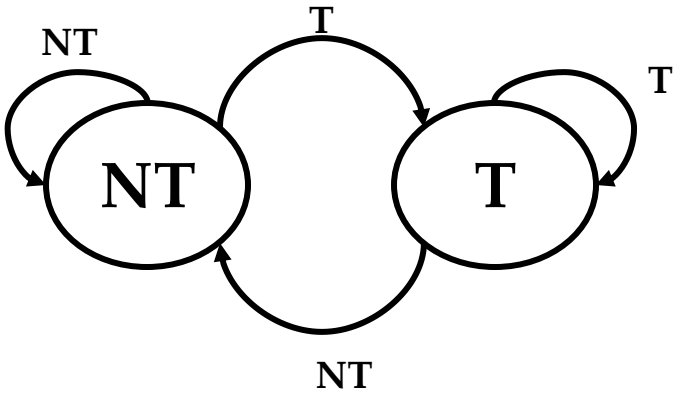
NT

1bit - BHT - Misprediction

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000

R1 is set to 0



Let us consider that the branch addresses do collide

LOOP2
100 iterations

LOOP
1000 iterations

1-BHT

T

1-BHT

NT

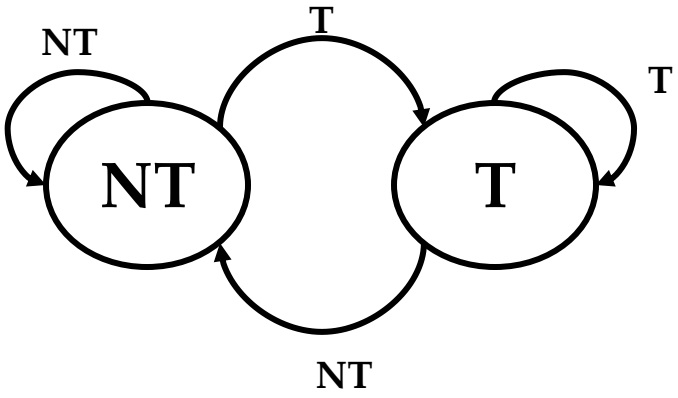
(1+1) * (1000-1) + 1

1bit - BHT - Misprediction

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000

R1 is set to 0



Let us consider that the branch addresses do collide

LOOP2
100 iterations

LOOP
1000 iterations

1-BHT

T

1-BHT

NT

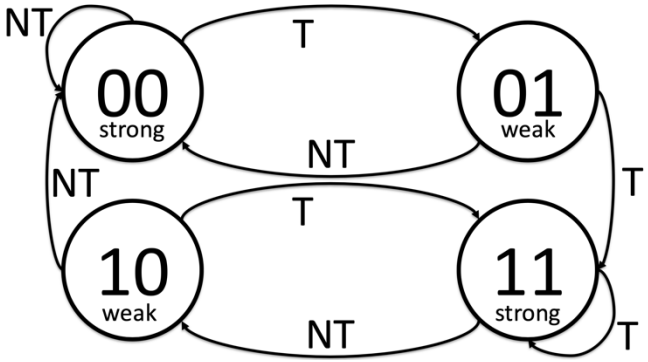
(1+1) * (1000-1) + 1

1 + (1+1) * (1000-1) + 1

2bit - BHT - Not Collide

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000
R1 is set to 0



Let us consider that the branch addresses do not collide

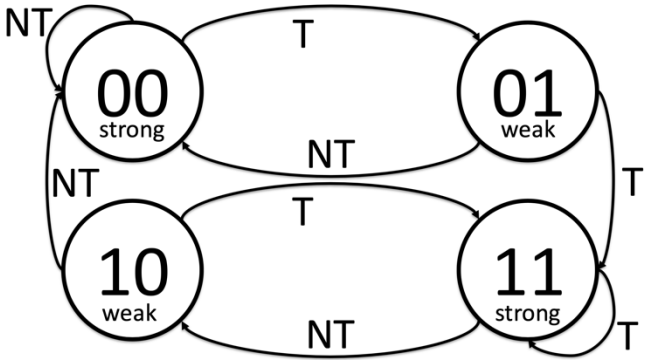
1-BHT

LOOP:	11
LOOP2:	11

2bit - BHT - Not Collide

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000
R1 is set to 0



Let us consider that the branch addresses do not collide

1-BHT

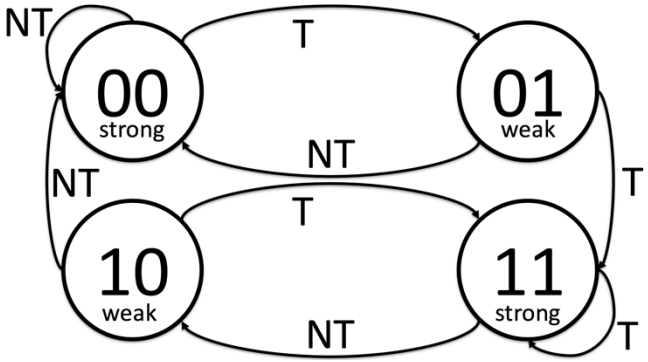
LOOP:	11
LOOP2:	11

1 +
(1000) * 1

2bit - BHT - Not Collide

```
LOOP:   LD      F1      0       R0
        ADDD    F2      F1      F1
        ADDI    R1      R1      100
LOOP2:  MULTD   F2      F2      F1
        SUBI    R1      R1      1
        BNEZ    R1      LOOP2
        SUBI    R0      R0      2
        BNEZ    R0      LOOP
```

R0 is set to 2000
R1 is set to 0



Let us consider that the branch addresses do not collide

1-BHT

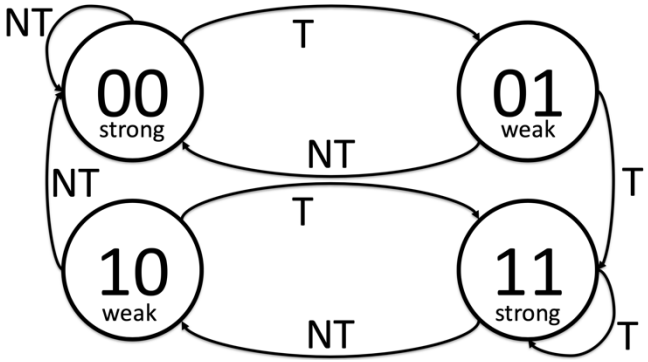
LOOP:	11
LOOP2:	11

1 +
(1000) * 1

2bit - BHT - Collide

```
LOOP:    LD      F1      0       R0
         ADDD    F2      F1      F1
         ADDI    R1      R1      100
LOOP2:   MULTD   F2      F2      F1
         SUBI    R1      R1      1
         BNEZ    R1      LOOP2
         SUBI    R0      R0      2
         BNEZ    R0      LOOP
```

R0 is set to 2000
R1 is set to 0



Let us consider that the branch addresses do collide

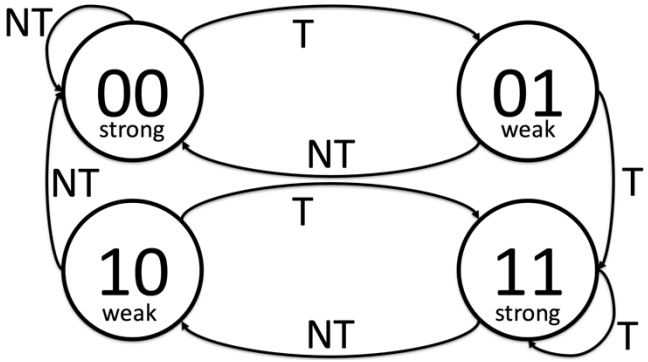
1-BHT

11

2bit - BHT - Collide

```
LOOP:    LD      F1      0       R0
         ADDD    F2      F1      F1
         ADDI    R1      R1      100
LOOP2:   MULTD   F2      F2      F1
         SUBI    R1      R1      1
         BNEZ    R1      LOOP2
         SUBI    R0      R0      2
         BNEZ    R0      LOOP
```

R0 is set to 2000
R1 is set to 0



Let us consider that the branch addresses do collide

1-BHT

11

1 +
(1000) * 1

SUMMARY

Assumption: NO collision

WORST CASES

1-BHT

LOOP:	NT
LOOP2:	NT

1000*2 misprediction for LOOP2
2 misprediction for LOOP.

2-BHT

LOOP:	NT_{strong}
LOOP2:	NT_{strong}

3+(1000-1)*1 misprediction for LOOP2
3 misprediction for LOOP.

BEST CASES

1-BHT

LOOP:	T
LOOP2:	T

1 + (1000-1) * 2 misprediction for LOOP2
1 for LOOP

2-BHT

LOOP:	T_{strong}
LOOP2:	T_{strong}

1000*1 misprediction for LOOP2
1 for LOOP

SUMMARY

Assumption: NO collision

WORST CASES

1-BHT

LOOP:
LOOP2:

1000*2 misprediction for LOOP2
2 misprediction for LOOP.

2-BHT

LOOP
LOOP2:

3+(1000-1)*1 misprediction for LOOP2
3 misprediction for LOOP.

WORST 2BHT better than BEST 1BHT

BEST CASES

1-BHT

LOOP: T
LOOP2: T

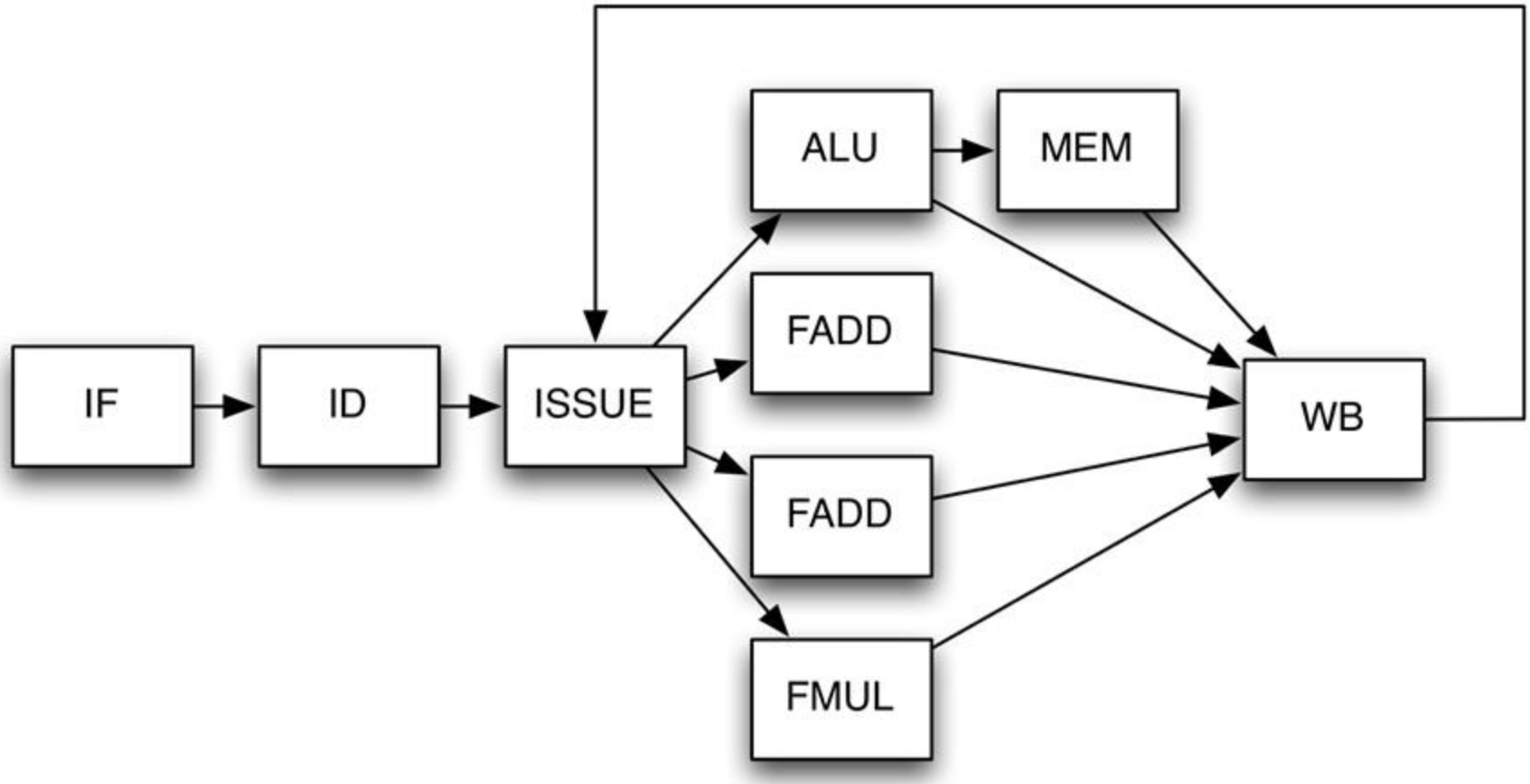
1 + (1000-1) * 2 misprediction for LOOP2
1 for LOOP

2-BHT

LOOP: T_{strong}
LOOP2: T_{strong}

1000*1 misprediction for LOOP2
1 for LOOP

Exe1: Complex Pipeline



Recall: Floating Point Arithmetic

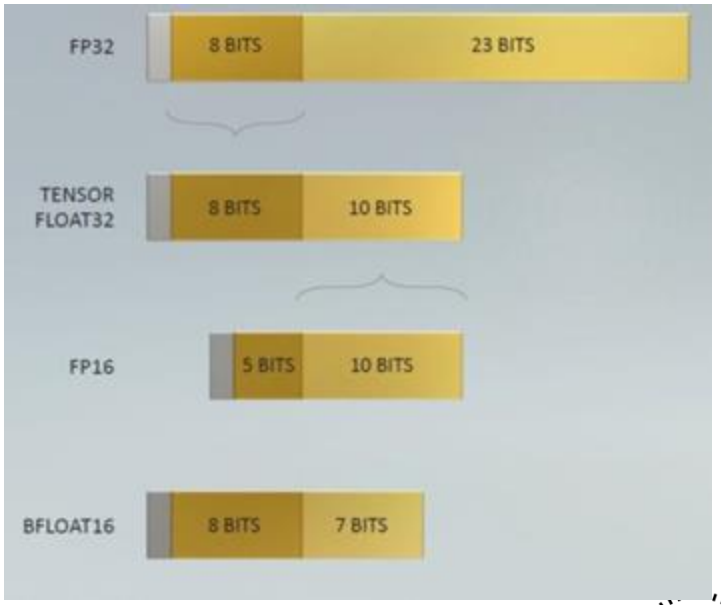
Real numbers such as:

pi → 3.14159265...

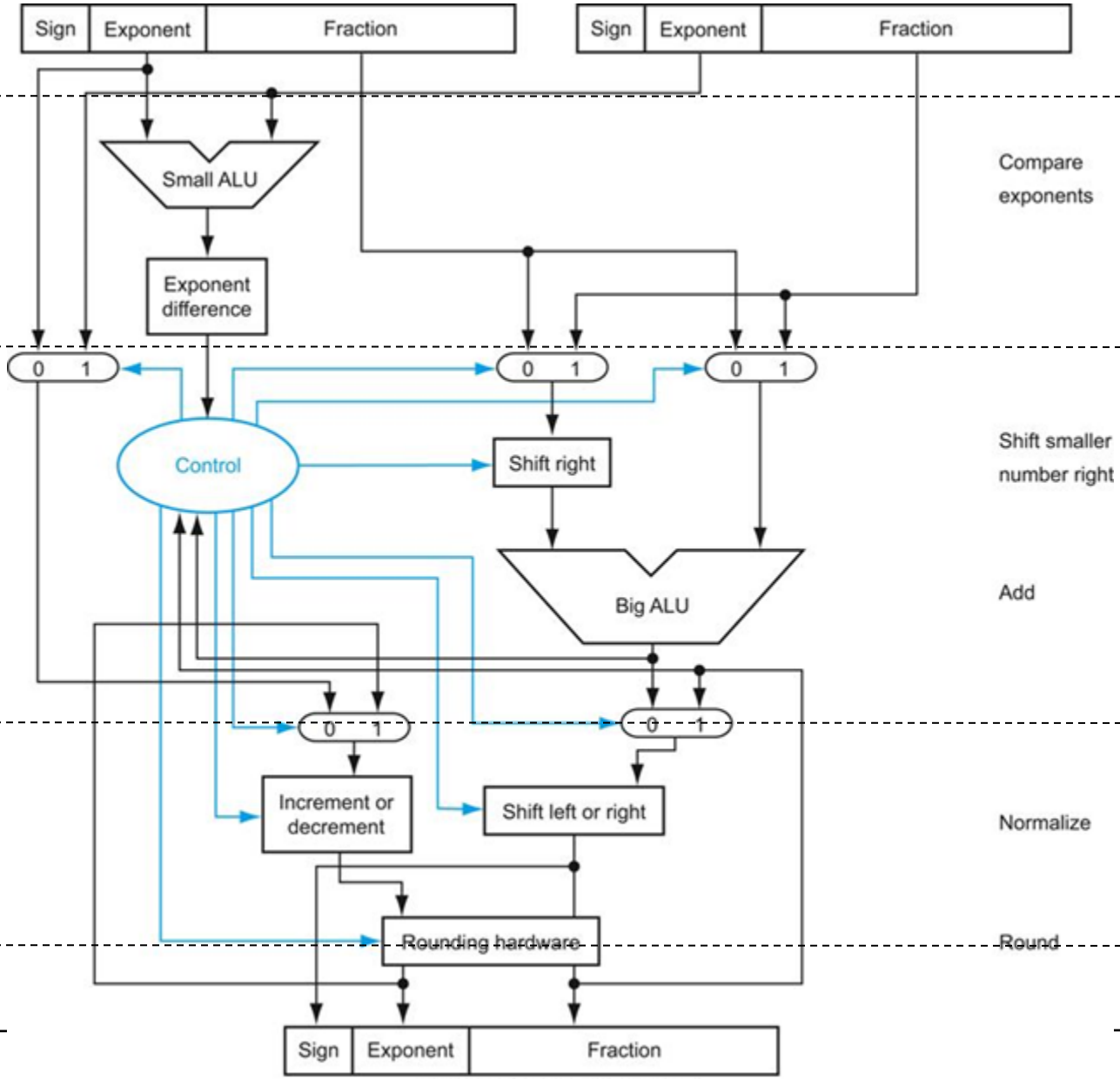
e → 2.81828...

seconds in a nanosecond → $1.0 * 10^{-9}$ or 0.000000001

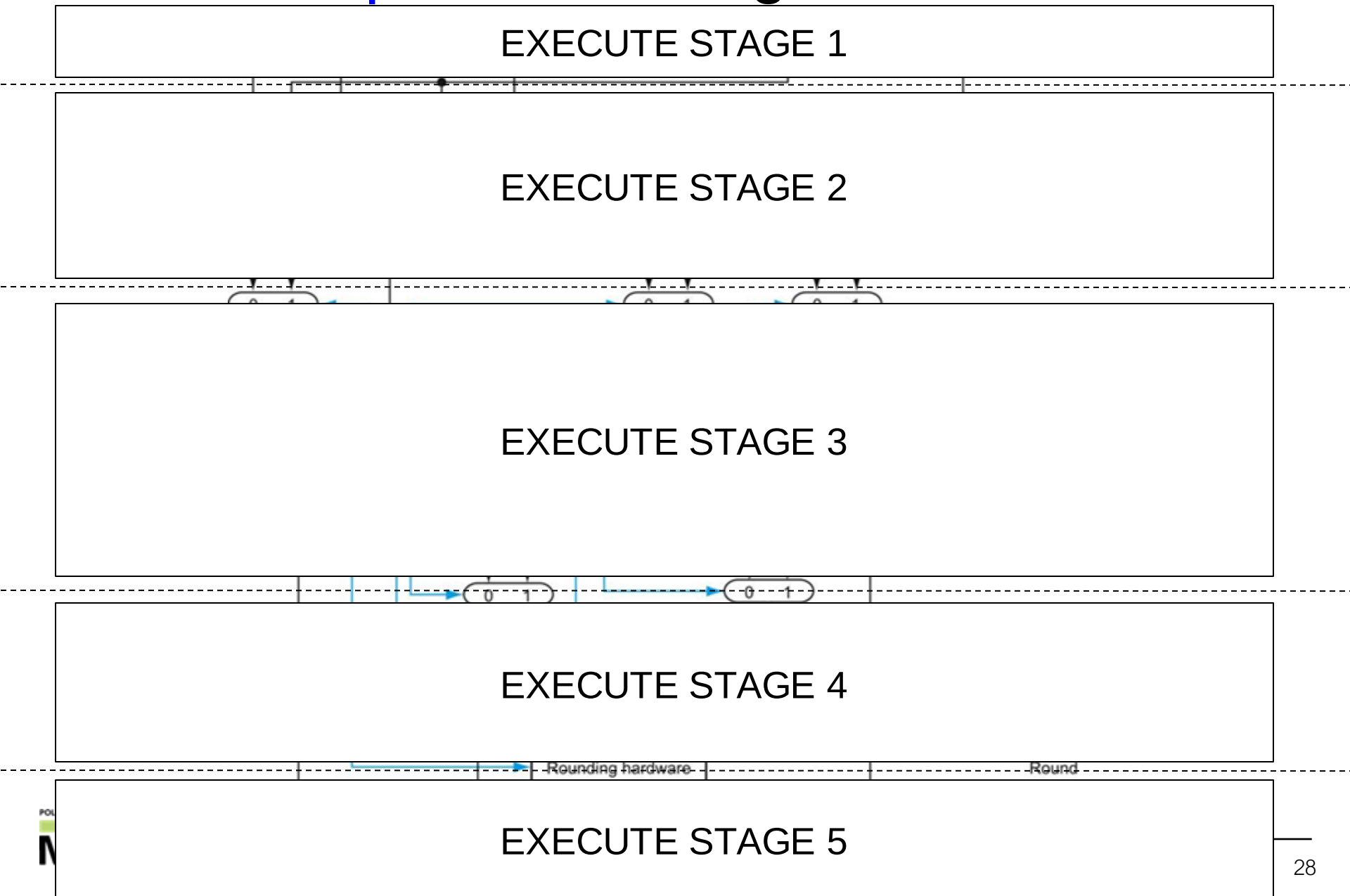
seconds in a typical century → $3.15576 * 10^{9}$ or 3,155,760,000



Example of Floating Point Adder



Example of Floating Point Adder



Recall: Three Classes of Hazards

Structural Hazards: Attempt to use the same resource from different instructions simultaneously

Example: Single memory for instructions and data

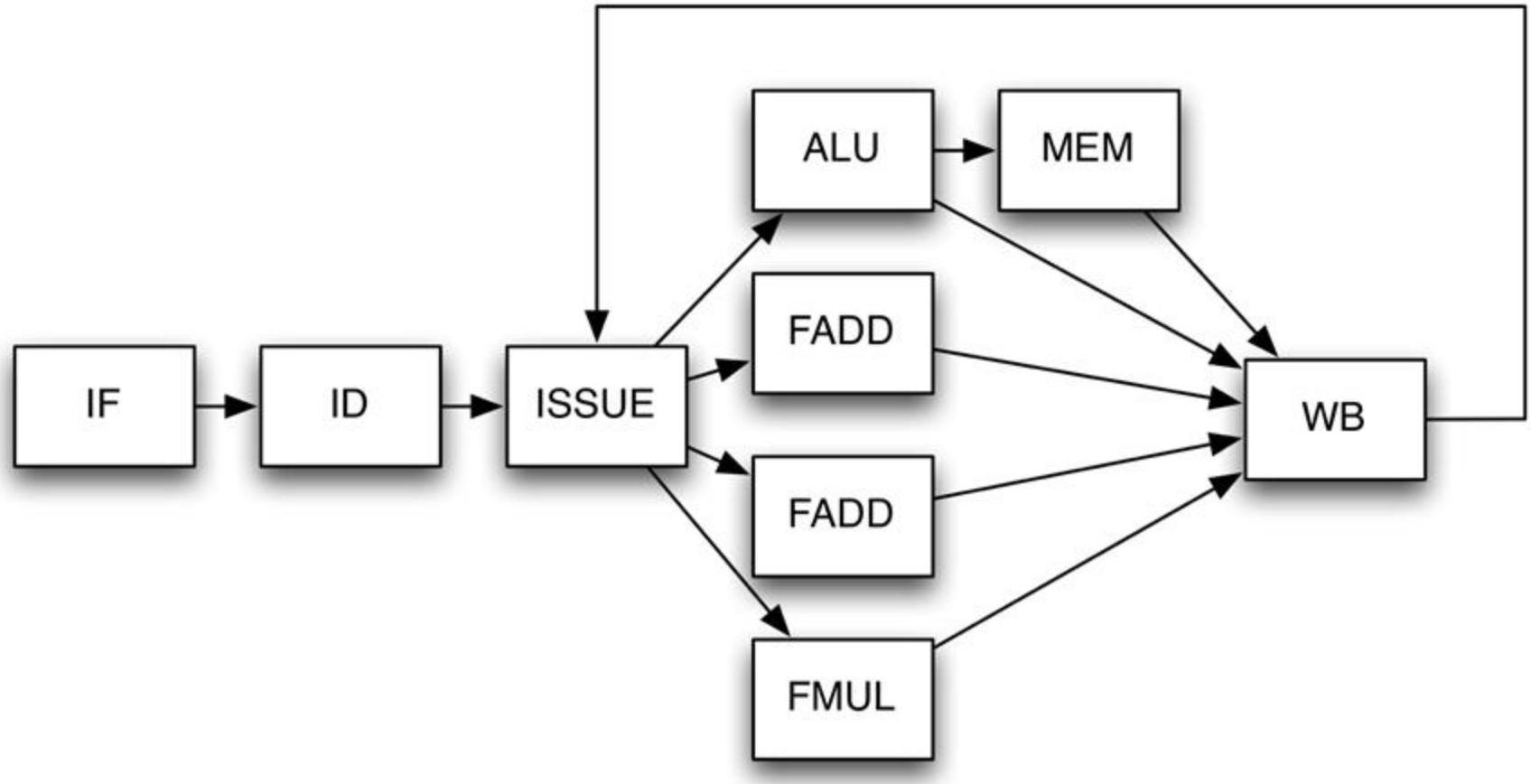
Data Hazards: Attempt to use a result before it is ready

Example: Instruction depending on a result of a previous instruction still in the pipeline

Control Hazards: Attempt to make a decision on the next instruction to execute before the condition is evaluated

Example: Conditional branch execution

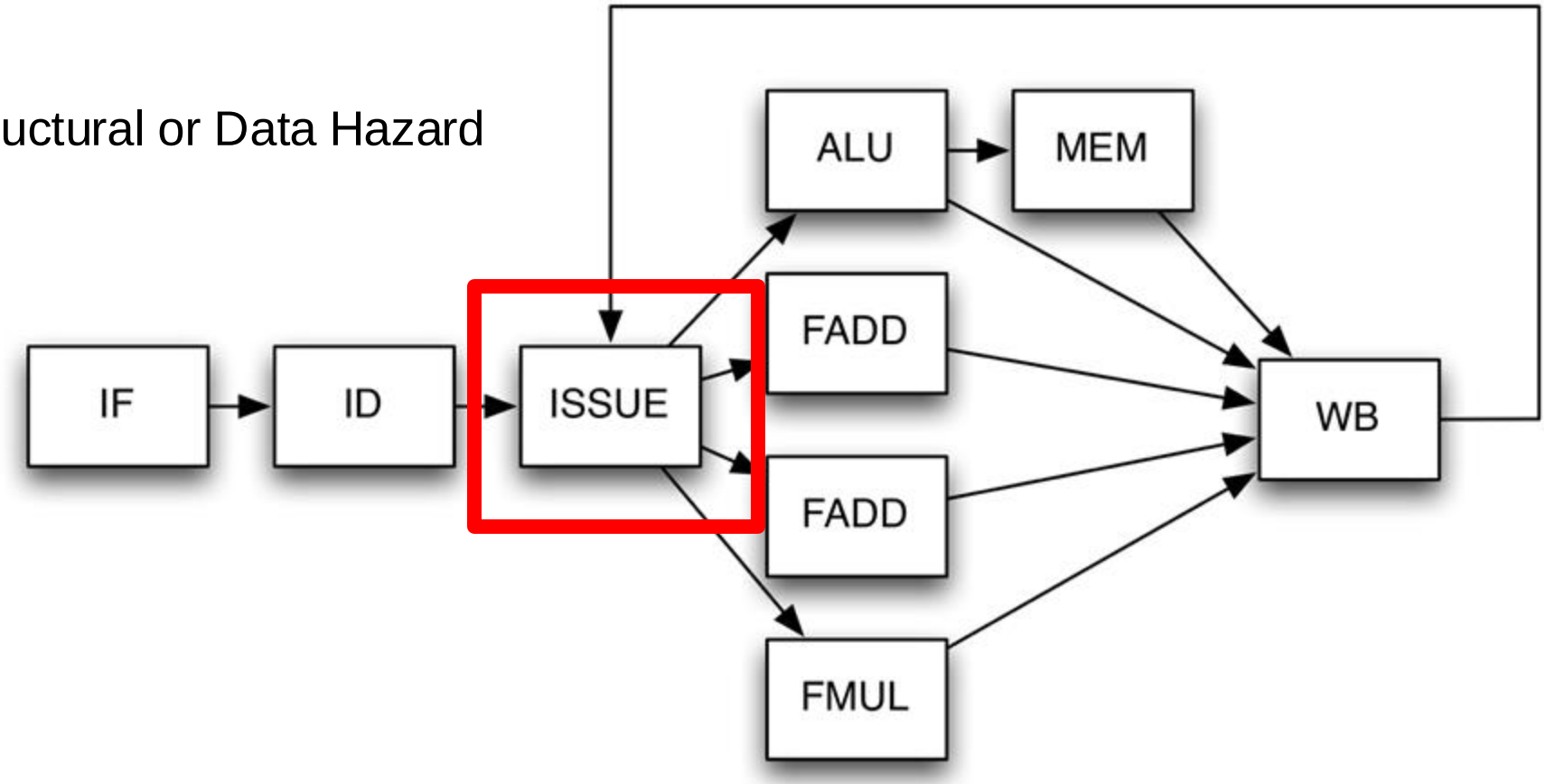
Exe1: Complex Pipeline



Exe1: Complex Pipeline

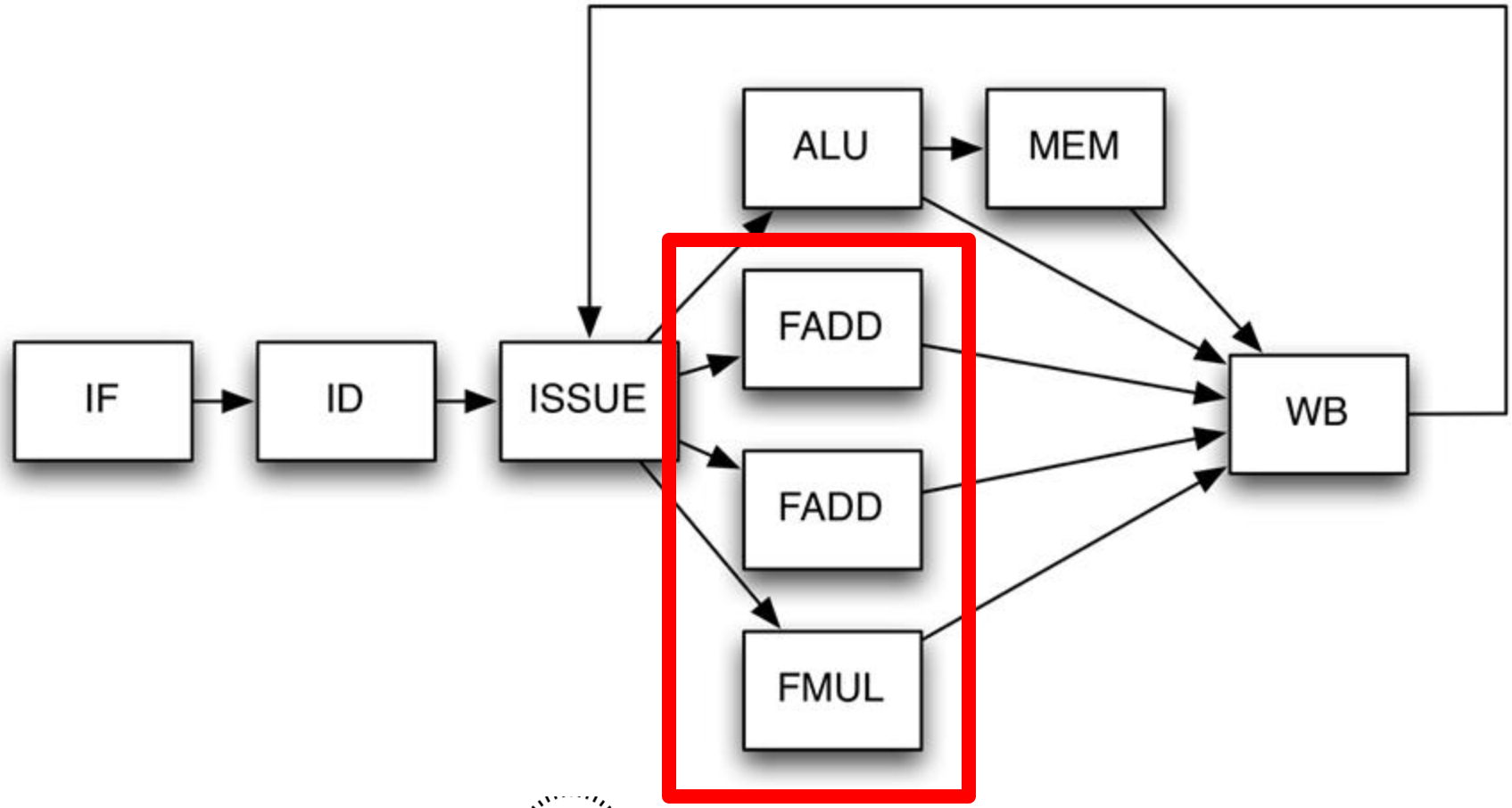
Is the required function unit available?
Is the input data available? ---> RAW?
Is it safe to write the destination? ---> WAR? WAW?
Is there a structural conflict at the WB stage?

Structural or Data Hazard



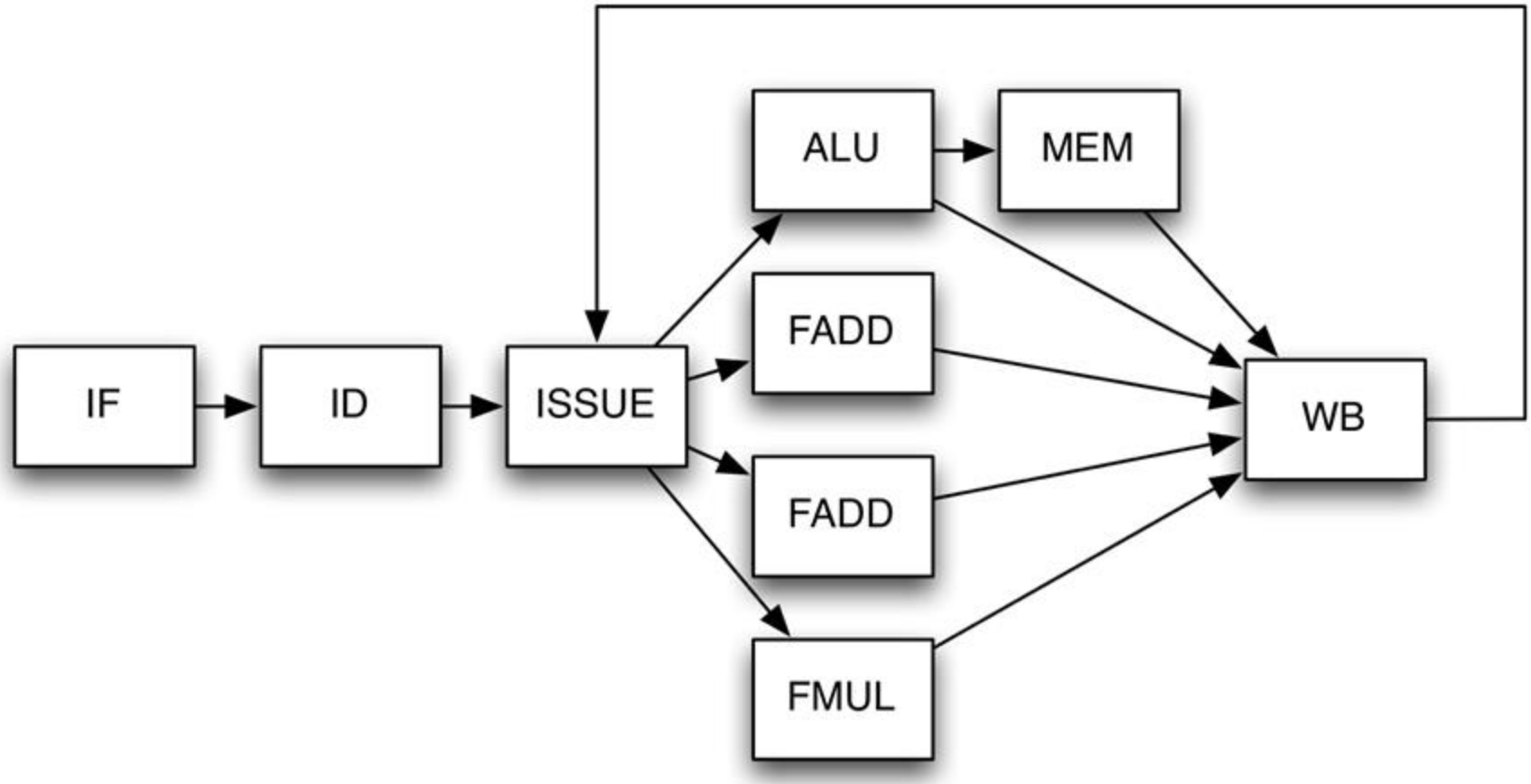
Exe1: Complex Pipeline

Multicycle, Pipelined (?), Initiation Interval (?)



Exe1: Complex Pipeline

In this problem we will examine the execution of a code segment on the following single-issue out-of-order processor:



Recall Hazards and Dependencies (1/3)

1) RAW (READ AFTER WRITE) hazards: instruction n+1 tries to **read** a **source** register **before** the previous instruction n **has written it** in the RF

Caused by a "**dependence**" (in compiler nomenclature) This hazard results from an actual need for **communication**.

2) WAW (WRITE AFTER WRITE) hazards: Instruction n+1 tries to **write** a **destination** operand **before** it has been **written** by the previous instruction n → **write** operations executed in the **wrong order**

Called an "**output dependence**" by compiler writers. This also results from the **reuse of name** "r1".

3) WAR (WRITE AFTER READ) hazards: Instruction n+1 tries to **write** a **destination** operand **before** it has **been read** from the previous instruction n → instruction n r**eads the wrong value**

Called an "**anti-dependence**" by compiler writers. This results from **reuse of the name** "r1".11

Recall Hazards and Dependencies (2/3)

		dest	src1	src2
I_1	DIVD	f6,	f6,	f4
I_2	LD	f2,	45(r3)	
I_3	MULTD	f0,	f2,	f4
I_4	DIVD	f8,	f6,	f2
I_5	SUBD	f10,	f0,	f6
I_6	ADDD	f6,	f8,	f2

RAW Hazards *WAW Hazards* *WAR Hazards*

Recall Hazards and Dependencies (3/3)

Data-dependence

$r_3 \leftarrow (r_1)$ op (r_2) Read-after-Write
$r_5 \leftarrow (r_3)$ op (r_4) (RAW) hazard

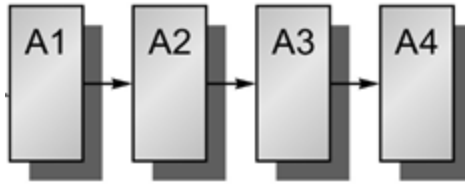
Anti-dependence

$r_3 \leftarrow (r_1)$ op (r_2) Write-after-Read
$r_1 \leftarrow (r_4)$ op (r_5) (WAR) hazard

Output-dependence

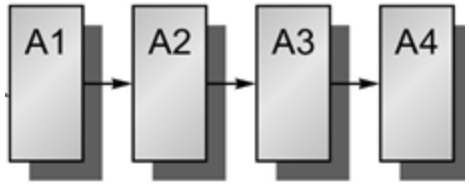
$r_3 \leftarrow (r_1)$ op (r_2) Write-after-Write
$r_3 \leftarrow (r_6)$ op (r_7) (WAW) hazard

You can assume that



- All functional units are pipelined
- ALU operations take 1 cycle
- Memory operations take 3 cycles (includes time in ALU)
- Floating-point add instructions take 3 cycles
- Floating-point multiply instructions take 5 cycles
- There is no register renaming. No forwarding
- Instructions are fetched, decoded and issued in order
- The ISSUE stage is a buffer of limited length that holds instructions waiting to start execution
- An instruction will only enter the issue stage if it does not cause a WAR or WAW hazard
- Only one instruction can be issued at a time, and in the case multiple instructions are ready, the oldest one will go first
- Program Counter calculation for branches and jumps has been anticipated in the ISSUE stage

You can assume that



- All functional units are pipelined
- ALU operations take 1 cycle
- Memory operations take 3 cycles (includes time in ALU)
- Floating-point add instructions take 3 cycles
- Floating-point multiply instructions take 5 cycles
- There is no register renaming. No forwarding
- Instructions are fetched, decoded and issued in order
- The ISSUE stage is a buffer of limited length that holds instructions waiting to start execution
- An instruction will only enter the issue stage if it does not cause a WAR or WAW hazard
- Only one instruction can be issued at a time, and in the case multiple instructions are ready, the oldest one will go first
- Program Counter calculation for branches and jumps has been anticipated in the ISSUE stage

Code

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

Code and Architecture

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

Conflicts

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW `I1-I2 $F3`

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW **I1-I2 $F3**

RAW **I3-I6 $F5**

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW I1-I2 $F3

RAW I4-I5 $R0

RAW I3-I6 $F5

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW I1-I2 $F3

RAW I4-I5 $R0

RAW I3-I6 $F5

RAW I5-I6 $F3

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW I1-I2 $F3

WAW I1-I5 $F3

RAW I4-I5 $R0

RAW I3-I6 $F5

RAW I5-I6 $F3

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW I1-I2 $F3

WAW I1-I5 $F3

RAW I4-I5 $R0

RAW I3-I6 $F5

WAW I2-I6 $F2

RAW I5-I6 $F3

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

Conflicts

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3
RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

Is this a possible execution? (1/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
I1 lw.d $F3,B($R0)			F	D	IS	E1	E2	E3	W												
I2 add.d $F2,$F2,$F3				F	D	IS	E1	E2	E3	W											
I3 mul.d $F5,$F4,$F4					F	D	IS	E1	E2	E3	E4	E5	W								
I4 addi $R0,$R0,8	F	D	IS	E	W																
I5 lw.d $F3,B($R0)		F	D	IS	E1	E2	E3	W													
I6 add.d $F2,$F3,$F5							F	D	IS	E1	E2	E3	W								

Is this a possible execution? (1/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Fetch, Decode, Issue are not in order

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
I1 lw.d $F3,B($R0)			F	D	IS	E1	E2	E3	W												
I2 add.d $F2,$F2,$F3				F	D	IS	E1	E2	E3	W											
I3 mul.d $F5,$F4,$F4					F	D	IS	E1	E2	E3	E4	E5	W								
I4 addi $R0,$R0,8	F	D	IS	E	W																
I5 lw.d $F3,B($R0)		F	D	IS	E1	E2	E3	W													
I6 add.d $F2,$F3,$F5							F	D	IS	E1	E2	E3	W								

Is this a possible execution? (2/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W														
I2 add.d $F2,$F2,$F3		F	D	IS	E1	E2	E3	W													
I3 mul.d $F5,$F4,$F4			F	D	IS	E1	E2	E3	E4	E5	W										
I4 addi $R0,$R0,8				F	D	IS	E	W													
I5 lw.d $F3,B($R0)					F	D	IS	E1	E2	E3	W										
I6 add.d $F2,$F3,$F5						F	D	IS	E1	E2	E3	W									

Is this a possible execution? (2/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Data hazards

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W														
I2 add.d $F2,$F2,$F3		F	D	IS	E1	E2	E3	W													
I3 mul.d $F5,$F4,$F4			F	D	IS	E1	E2	E3	E4	E5	W										
I4 addi $R0,$R0,8				F	D	IS	E	W													
I5 lw.d $F3,B($R0)					F	D	IS	E1	E2	E3	W										
I6 add.d $F2,$F3,$F5						F	D	IS	E1	E2	E3	W									

Is this a possible execution? (3/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D s	D s	D s	D	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F s	F s	F s	F	D	IS	E	W												
I5 lw.d $F3,B($R0)								F	D s	D s	D	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5									F s	F s	F	D	IS s	IS s	IS s	IS	E1	E2	E3	W			

Is this a possible execution? (3/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Concurrent write

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D s	D s	D s	D	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F s	F s	F s	F	D	IS	E	W												
I5 lw.d $F3,B($R0)								F	D s	D s	D	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5									F s	F s	F	D	IS s	IS s	IS s	IS	E1	E2	E3	W			

Is this a possible execution? (4/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D s	D s	D s	D	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F s	F s	F s	F	D	IS	E	E s	W											
I5 lw.d $F3,B($R0)								F	D s	D	IS s	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5									F s	F	D s	D	IS s	IS s	IS s	IS	E1	E2	E3	W			

Is this a possible execution? (4/4)

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

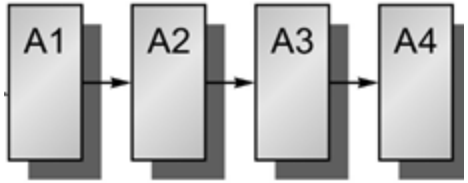
RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Correct execution

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D s	D s	D s	D	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F s	F s	F s	F	D	IS	E	E s	W											
I5 lw.d $F3,B($R0)								F	D s	D	IS s	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5									F s	F	D s	D	IS s	IS s	IS s	IS	E1	E2	E3	W			

What if we change the buffer dimension?



- All functional units are pipelined
- ALU operations take 1 cycle
- Memory operations take 3 cycles (includes time in ALU)
- Floating-point add instructions take 3 cycles
- Floating-point multiply instructions take 5 cycles
- There is no register renaming. No forwarding
- Instructions are fetched, decoded and issued in order
- The issue stage is a buffer of unlimited length that holds instructions waiting to start execution
- An instruction will only enter the issue stage if it does not cause a WAR or WAW hazard
- Only one instruction can be issued at a time, and in the case multiple instructions are ready, the oldest one will go first
- Program Counter calculation for branches and jumps has been anticipated in the ISSUE stage

How will this change the execution?

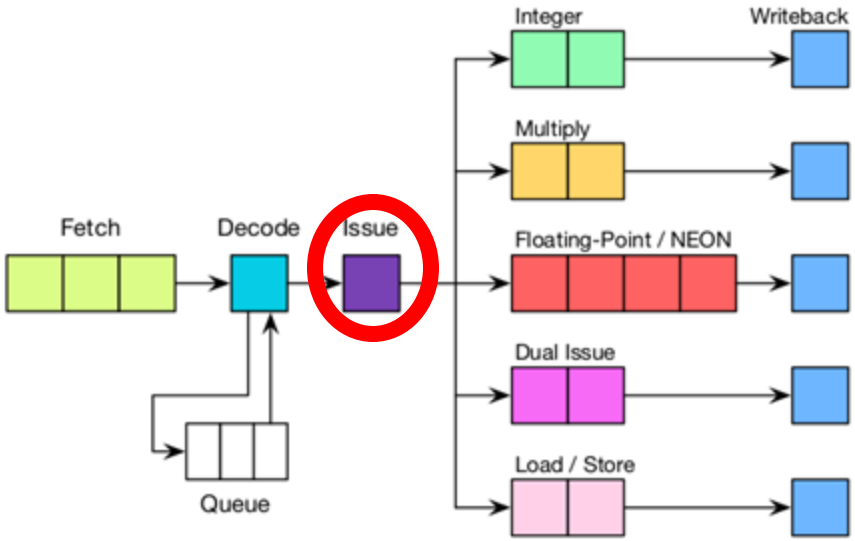
RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

WAR I1-I6 $R0
RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

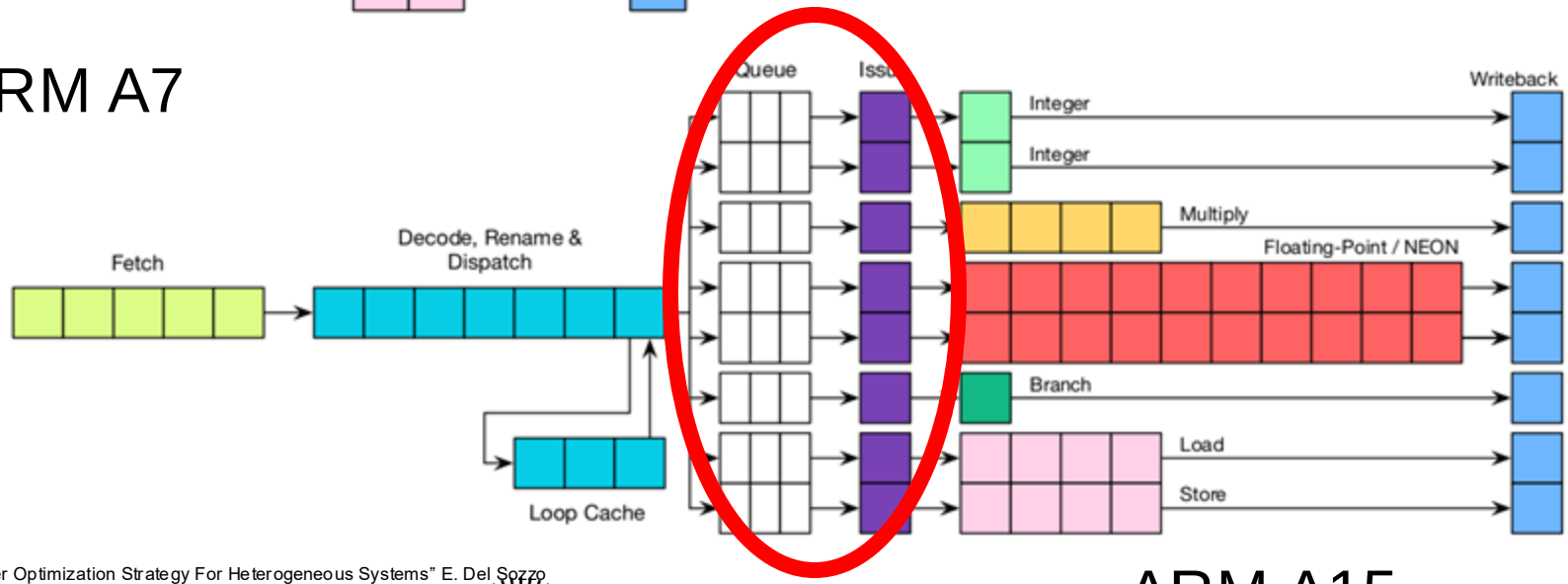
ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D s	D s	D s	D	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F s	F s	F s	F	D	IS	E	E s	W											
I5 lw.d $F3,B($R0)								F	D s	D	IS s	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5									F s	F	D s	D	IS s	IS s	IS s	IS	E1	E2	E3	W			

An Example of Complex Pipeline (ARM)



ARM A7



ARM A15

Source: "Workload-Aware Power Optimization Strategy For Heterogeneous Systems" E. Del Sozzo

How will this change the execution?

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

WAR I1-I6 $R0
RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D s	D s	D s	D	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F s	F s	F s	F	D	IS	E	E s	W											
I5 lw.d $F3,B($R0)								F	D s	D	IS s	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5									F s	F	D s	D	IS s	IS s	IS s	IS	E1	E2	E3	W			

How will this change the execution?

RAW I1-I2 $F3
WAR I1-I4 $R0
WAW I1-I5 $F3
WAR I2-I5 $F3

WAR I1-I6 $R0
RAW I4-I5 $R0
RAW I3-I6 $F5
WAW I2-I6 $F2
RAW I5-I6 $F3

ALU OP: 1cc
MEM OP: 3cc
FP ADD: 3cc
FP MULT: 5cc

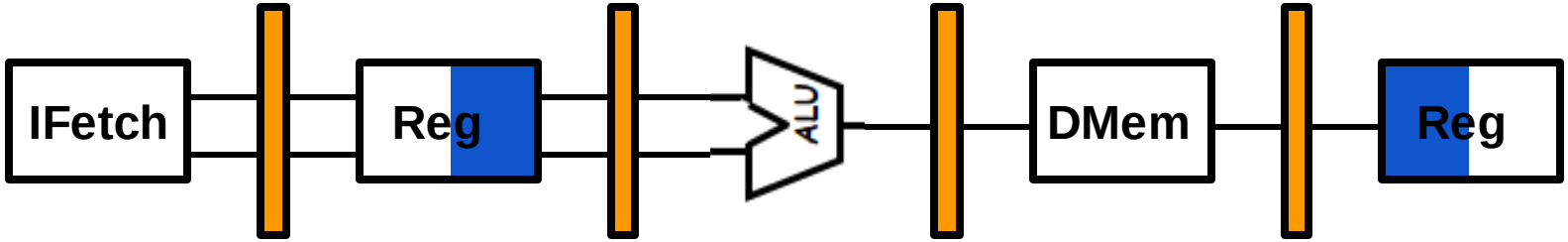
Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23
I1 lw.d $F3,B($R0)	F	D	IS	E1	E2	E3	W																
I2 add.d $F2,$F2,$F3		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W												
I3 mul.d $F5,$F4,$F4			F	D	IS s	IS s	IS s	IS	E1	E2	E3	E4	E5	W									
I4 addi $R0,$R0,8				F	D s	D	IS s	IS s	IS	E	E s	W											
I5 lw.d $F3,B($R0)					F s	F	D s	D s	D s	D	IS s	IS	E1	E2	E3	W							
I6 add.d $F2,$F3,$F5							F s	F s	F s	F	D	IS s	IS s	IS s	IS s	IS	E1	E2	E3	W			

Exe 3: Simple Pipelining

Exe 3 Simple Pipelining : the Code

```
I1:  addi $s3, $s2, 2
I2:  add $s5, $s4, $s3
I3:  sw $s5, 4($s3)
I4:  sub $s7, $s5, $s6
I5:  lw $s6, 4($s7)
```

Exe 3: Simple Pipelining: the Architecture



Exe 3.1 Simple Pipelining : Conflicts

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	**F**	**D**	**E**	**M**	**W**										
I2:	`add $s5, $s4, $s3`		**F**	**D**	**E**	**M**	**W**									
I3:	`sw $s5, 4($s3)`			**F**	**D**	**E**	**M**	**W**								
I4:	`sub $s7, $s5, $s6`				**F**	**D**	**E**	**M**	**W**							
I5:	`lw $s6, 4($s7)`					**F**	**D**	**E**	**M**	**W**						

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : solve as is

Istr	CK1	CK2	CK3	CK4	CK5	CK6	CK7	CK8	CK9	CK10	CK11
I1											
I2											
I3											
I4											
I5											
Istr	CK12	CK13	CK14	CK15	CK16	CK17	CK18	CK19	CK20	CK21	CK22
I1											
I2											
I3											
I4											
I5											

```
I1:  addi $s3, $s2, 2
I2:  sub $s4, $s3, $s1
I3:  add $s5, $s4, $s1
I4:  lw $s6, 4($s4)
I5:  sub $s7, $s4, $s6
```

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to **reduce the stalls**; Draw the pipeline schema showing all the data conflicts/dependencies.

Exe 3.3 Simple Pipelining : FWD Paths

Istr	CK1	CK2	CK3	CK4	CK5	CK6	CK7	CK8	CK9	CK10	CK11
I1											
I2											
I3											
I4											
I5											
Istr	CK12	CK13	CK14	CK15	CK16	CK17	CK18	CK19	CK20	CK21	CK22
I1											
I2											
I3											
I4											
I5											

```
I1:  addi $s3, $s2, 2
I2:  sub $s4, $s3, $s1
I3:  add $s5, $s4, $s1
I4:  lw $s6, 4($s4)
I5:  sub $s7, $s4, $s6
```

Exe 3.1 Simple Pipelining : Conflicts

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	**F**	**D**	**E**	**M**	**W**										
I2:	`add $s5, $s4, $s3`		**F**	**D**	**E**	**M**	**W**									
I3:	`sw $s5, 4($s3)`			**F**	**D**	**E**	**M**	**W**								
I4:	`sub $s7, $s5, $s6`				**F**	**D**	**E**	**M**	**W**							
I5:	`lw $s6, 4($s7)`					**F**	**D**	**E**	**M**	**W**						

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : Conflicts

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	F	D	E	M	W										
I2:	`add $s5, $s4, $s3`		F	D	E	M	W									
I3:	`sw $s5, 4($s3)`			F	D	E	M	W								
I4:	`sub $s7, $s5, $s6`				F	D	E	M	W							
I5:	`lw $s6, 4($s7)`					F	D	E	M	W						

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : Hazards

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	F	D	E	M	W										
I2:	`add $s5, $s4, $s3`		F	D	E	M	W									
I3:	`sw $s5, 4($s3)`			F	D	E	M	W								
I4:	`sub $s7, $s5, $s6`				F	D	E	M	W							
I5:	`lw $s6, 4($s7)`					F	D	E	M	W						

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : solve as is

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										
3	sw $s5, 4($s3)			F	D	E	M	W									
4	sub $s7, $s5, $s6				F	D	E	M	W								
5	lw $s6, 4($s7)					F	D	E	M	W							

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.1 Simple Pipelining : solve as is

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D(s)	D(s)	D	E	M	W								
3	sw $s5, 4($s3)			F(s)	F(s)	F	D(s)	D(s)	D	E	M	W					
4	sub $s7, $s5, $s6						F(s)	F(s)	F	D	E	M	W				
5	lw $s6, 4($s7)									F	D(s)	D(s)	D	E	M	W	

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to **reduce the stalls**; Draw the pipeline schema showing all the data conflicts/dependencies.

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to reduce the stalls; Draw the pipeline schema showing all the data conflicts/dependencies.

```
I1:  addi $s3, $s2, 2
I2:  add $s5, $s4, $s3
I3:  sw $s5, 4($s3)
I4:  sub $s7, $s5, $s6
I5:  lw $s6, 4($s7)
```

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to reduce the stalls; Draw the pipeline schema showing all the data conflicts/dependencies.

```
I1:  addi $s3, $s2, 2
I2:  add $s5, $s4, $s3
I4:  sub $s7, $s5, $s6
I3:  sw $s5, 4($s3)
I5:  lw $s6, 4($s7)
```

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling RAW

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										
4	sub $s7, $s5, $s6			F	D	E	M	W									
3	sw $s5, 4($s3)				F	D	E	M	W								
5	lw $s6, 4($s7)					F	D	E	M	W							

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling Execution

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D(s)	D(s)	D	E	M	W								
4	sub $s7, $s5, $s6			F(s)	F(s)	F	D(s)	D(s)	D	E	M	W					
3	sw $s5, 4($s3)						F(s)	F(s)	F	D	E	M	W				
5	lw $s6, 4($s7)									F	D(s)	D	E	M	W		

RAW $s3 I1-I2

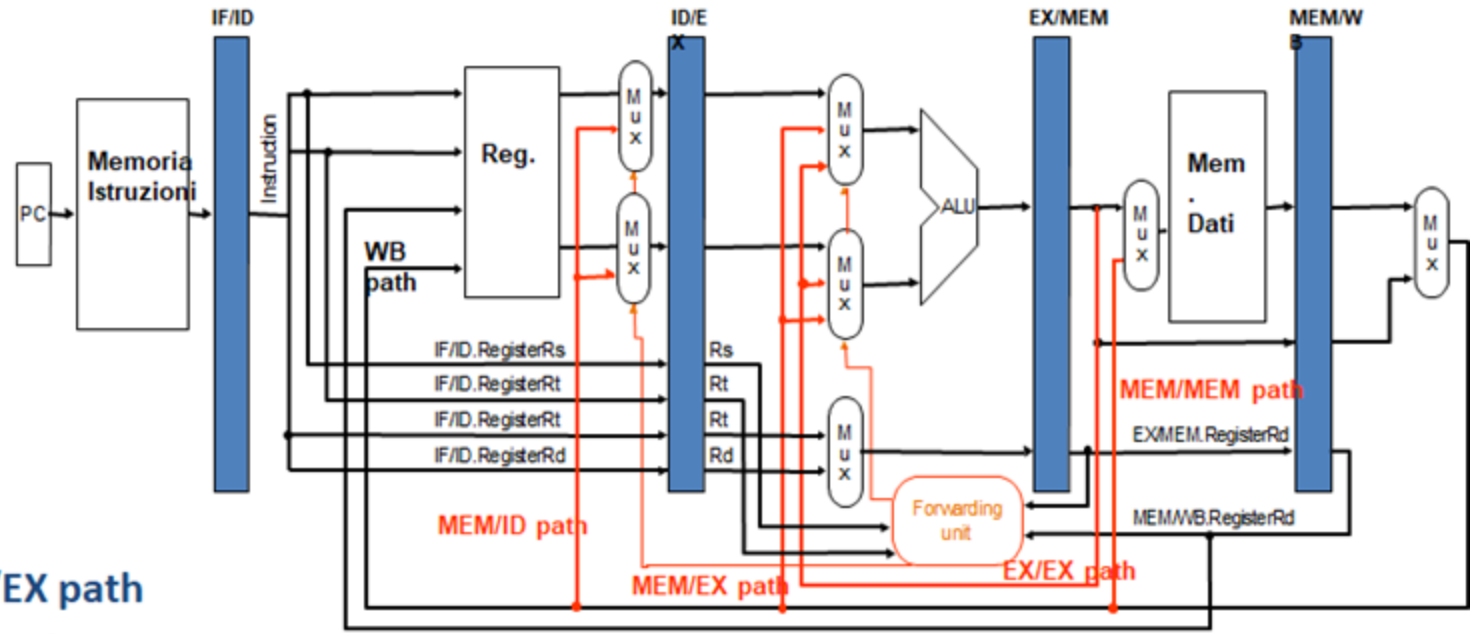
RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.3: Forwarding paths



- EX/EX path
- MEM/EX path
- MEM/ID path
- MEM/MEM path

The forwarding paths have been included in the pipeline. Start from the code in (a) and draw the pipeline schema showing all the forwarding paths that have to be used to solve the hazards.

Exe 3.3 Simple Pipelining : FWD Paths

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										
3	sw $s5, 4($s3)			F	D	E	M	W									
4	sub $s7, $s5, $s6				F	D	E	M	W								
5	lw $s6, 4($s7)					F	D	E	M	W							

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.3 Simple Pipelining : FWD Paths

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	FWD Path
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										EX-EX
3	sw $s5, 4($s3)			F	D	E	M	W									M-EX M-M
4	sub $s7, $s5, $s6				F	D	E	M	W								M-EX
5	lw $s6, 4($s7)					F	D	E	M	W							EX-EX

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Thank you for your attention

Questions?

Alessandro Verosimile <alessandro.verosimile@polimi.it>

Acknowledgements

Davide Conficconi, E. Del Sozzo, Marco D. Santambrogio, D. Sciuto

Part of this material comes from:

- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- News and paper cited throughout the lecture