POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

POLITECNICO MILANO 1863 NECST laboratory

Exercise Session 6

Scoreboard, Tomasulo, Dynamic Branch Prediction, (Extra:Simple Scheduling)

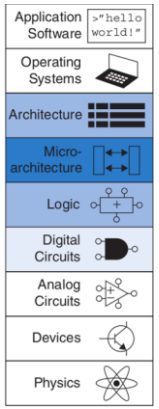
Advanced Computer Architectures

16th April 2025

Davide Conficconi ‹davide.conficconi@polimi.it›

POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

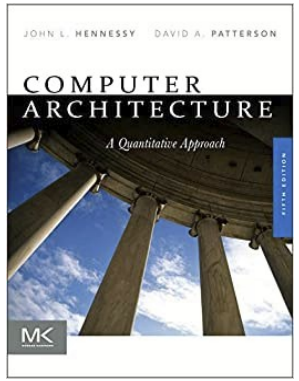
Recall: Material (EVERYTHING OPTIONAL)



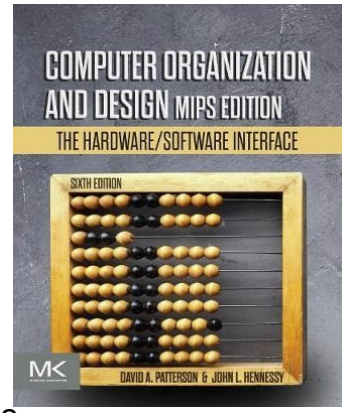
https://webeep.polimi.it/course/view.php?id=14754

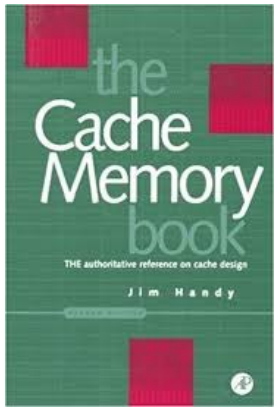
https://tinyurl.com/aca-grid25

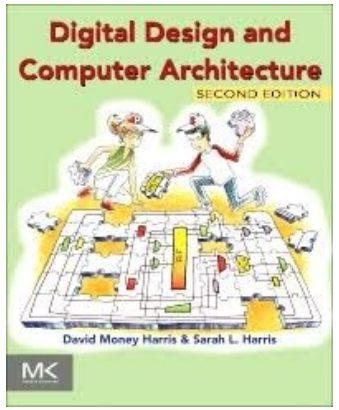
Textbook: Hennessy and Patterson, Computer Architecture: A Quantitative Approach

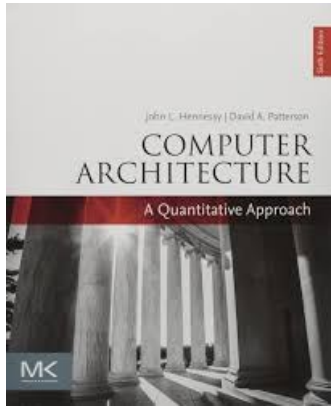


Other Interesting Reference

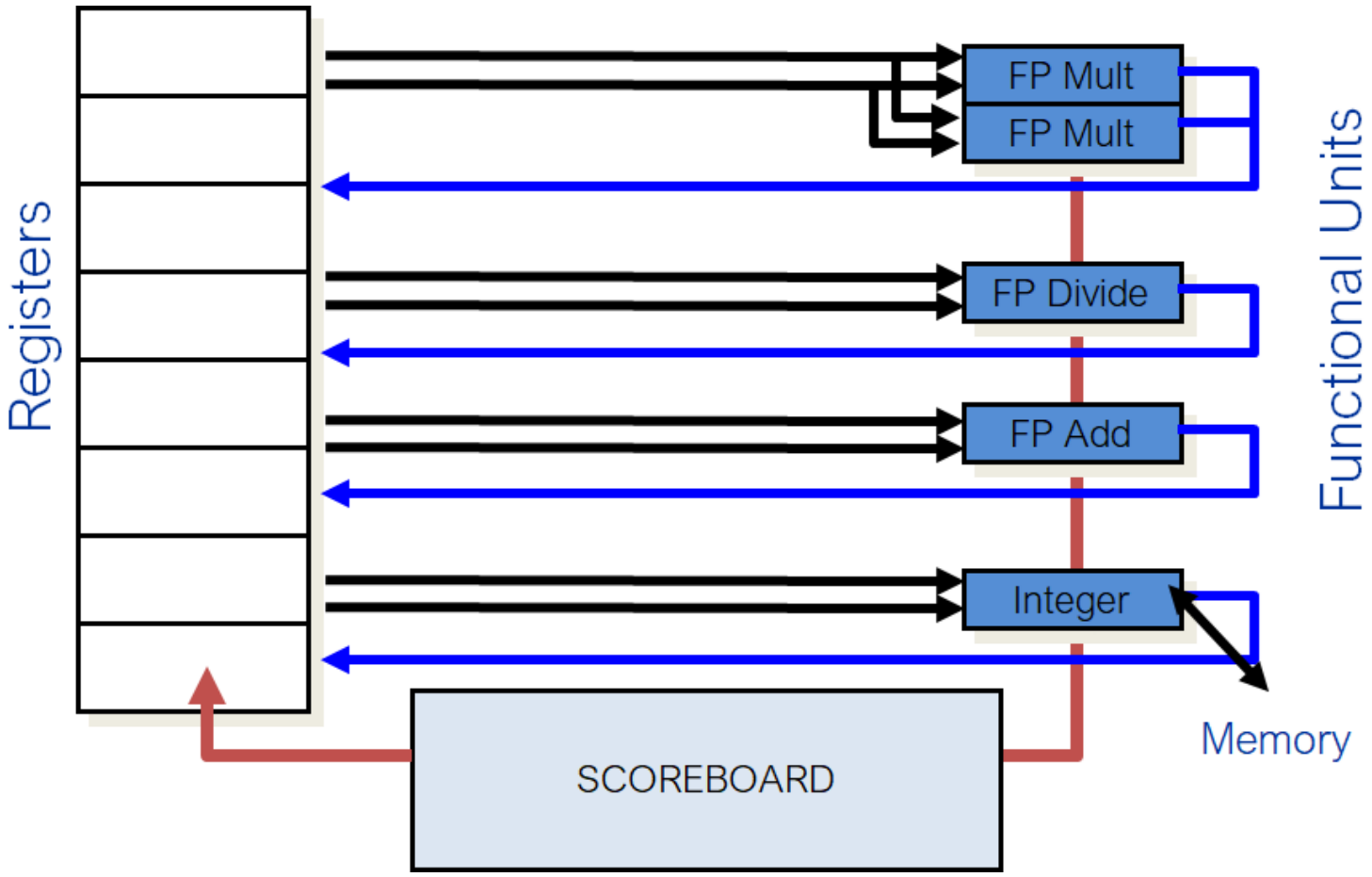








Exe Scoreboard



Parallel operation in the control data 6600

The Scoreboard pipeline

ISSUE	READ OPERAND	EXE COMPLETE	WB
Decode instruction;	Read operands;	Operate on operands;	Finish exec;
Structural FUs check; WAW checks	RAW check; WAR if need to read	Notify Scoreboard on completion;	WAR &Struct check (FUs will hold results); Can overlap issue/read&write 4 Structural Hazard;

Exe.1 The code

```
I1:  LD $F1, 0($R1)
I2:  FADD $F2, $F2, $F3
I3:  ADDI $R3, $R3, 8
I4:  LD $F4, 0(R2)
I5:  FADD $F5, $F4, $F2
I6:  FMULT $F6, $F1, $F4
I7:  ADDI $R5, $R5, 1
I8:  LD $R6, 0($R4)
I9:  SD $F6, 0($R5)
I10: SD $F5, 0($R6)
```

Exe.1 The conflicts

```
I1:  LD $F1, 0($R1)
I2:  FADD $F2, $F2, $F3
I3:  ADDI $R3, $R3, 8
I4:  LD $F4, 0(R2)
I5:  FADD $F5, $F4, $F2
I6:  FMULT $F6, $F1, $F4
I7:  ADDI $R5, $R5, 1
I8:  LD $R6, 0($R4)
I9:  SD $F6, 0($R5)
I10: SD $F5, 0($R6)
```

Exe.2 Scoreboard: ∃ a configuration?

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD $F1, 0($R1)	1	2	5	6
I2	FADD $F2, $F2, $F3	2	3	7	8
I3	ADDI $R3, $R3, 8	3	4	5	6
I4	LD $F4, 0(R2)	4	5	8	9
I5	FADD $F5, $F4, $F2	9	6	14	15
I6	FMULT $F6, $F1, $F4	8	10	14	16
I7	ADDI $R5, $R5, 1	5	8	9	10
I8	LD $R6, 0($R4)	6	9	12	13
I9	SD $F6, 0($R5)	9	11	14	17
I10	SD $F5, 0($R6)	10	14	17	18

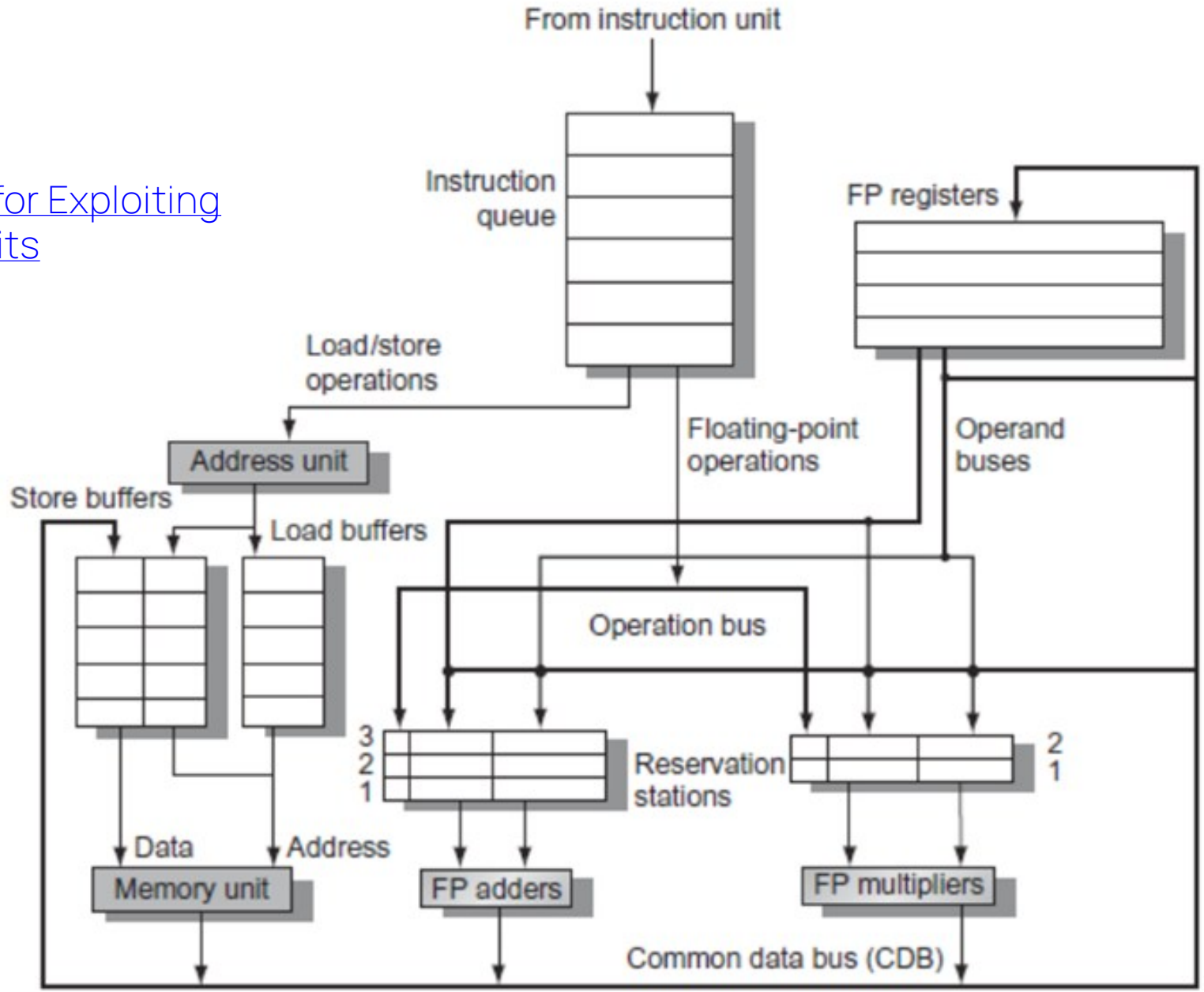
- Is there a “configuration” that can respect the shown execution?
- How many units? Which kind? What latency?

Exe.2 Scoreboard CC0

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD $F1, 0($R1)						
I2	FADD $F2, $F2, $F3						
I3	ADDI $R3, $R3, 8						
I4	LD $F4, 0(R2)						
I5	FADD $F5, $F4, $F2						
I6	FMULT $F6, $F1, $F4						
I7	ADDI $R5, $R5, 1						
I8	LD $R6, 0($R4)						
I9	SD $F6, 0($R5)						
I10	SD $F5, 0($R6)						

Exe Tomasulo

[An Efficient Algorithm for Exploiting Multiple Arithmetic Units]



Exe.1 The conflicts

```
I1:  LD $F1, 0($R1)
I2:  FADD $F2, $F2, $F3
I3:  ADDI $R3, $R3, 8
I4:  LD $F4, 0(R2)
I5:  FADD $F5, $F4, $F2
I6:  FMULT $F6, $F1, $F4
I7:  ADDI $R5, $R5, 1
I8:  LD $R6, 0($R4)
I9:  SD $F6, 0($R5)
I10: SD $F5, 0($R6)
```

Recall: the Tomasulo pipeline

ISSUE	EXECUTION	WRITE
Get Instruction from Queue and Rename Registers	Execute and Watch CDB;	Write on CDB;
Structural RSs check; WAW and WAR solved by Renaming (!!!in-order-issue!!!);	Check for Struct on FUs; RAW delaying; Struct check on CDB;	(FUs will hold results unless CDB free) RSs/FUs marked free

Exe.2 Tomasulo: ∃ a configuration?

Instruction	ISSUE	START EXE	WB
I1: LD $F1, 0($R1)	1	2	5
I2: FADD $F2, $F2, $F3	2	3	6
I3: ADDI $R3, $R3, 8	3	4	6
I4: LD $F4, 0(R2)	6	5	8
I5: FADD $F5, $F4, $F2	7	9	12
I6: FMULT $F6, $F1, $F4	8	9	13
I7: ADDI $R5, $R5, 1	9	10	11
I8: LD $R6, 0($R4)	10	9	14
I9: SD $F6, 0($R5)	11	14	17
I10: SD $F5, 0($R6)	12	15	18

- Is there a "configuration" that can respect the shown execution?
- How many units? Which kind? What latency? How many Reservation Stations?

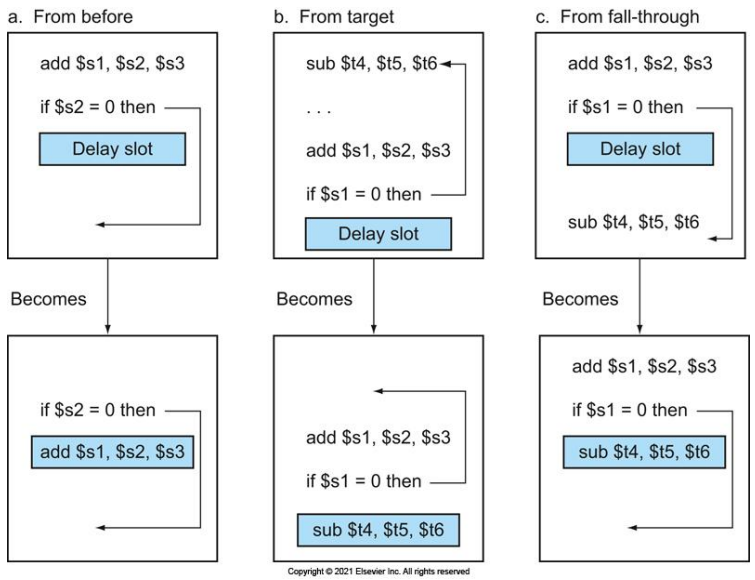
Exe.3 Tomasulo CC0

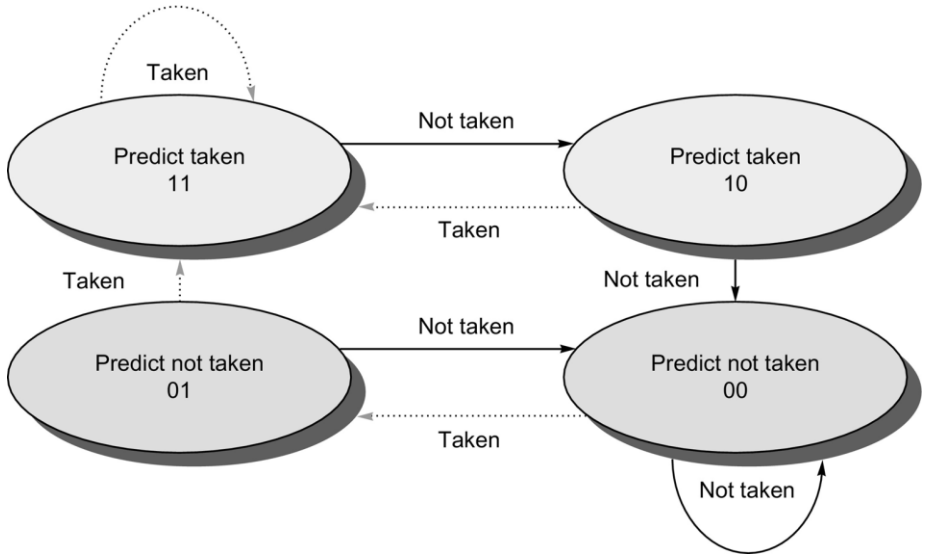
Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1: LD $F1, 0($R1)						
I2: FADD $F2, $F2, $F3						
I3: ADDI $R3, $R3, 8						
I4: LD $F4, 0(R2)						
I5: FADD $F5, $F4, $F2						
I6: FMULT $F6, $F1, $F4						
I7: ADDI $R5, $R5, 1						
I8: LD $R6, 0($R4)						
I9: SD $F6, 0($R5)						
I10: SD $F5, 0($R6)						

Prediction

[Branch vanguard: decomposing branch functionality into prediction and resolution instructions]

[The IBM z15 High Frequency Mainframe Branch Predictor]





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 1, R1 is set to 300)

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        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 F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

How many iterations?

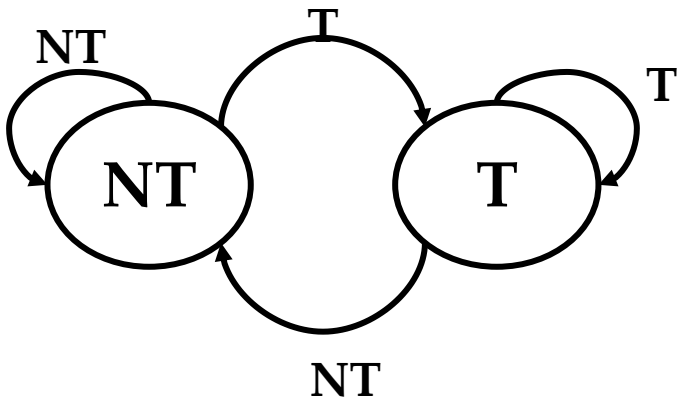
R0 is set to 1
R1 is set to 300

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

1bit - BHT

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

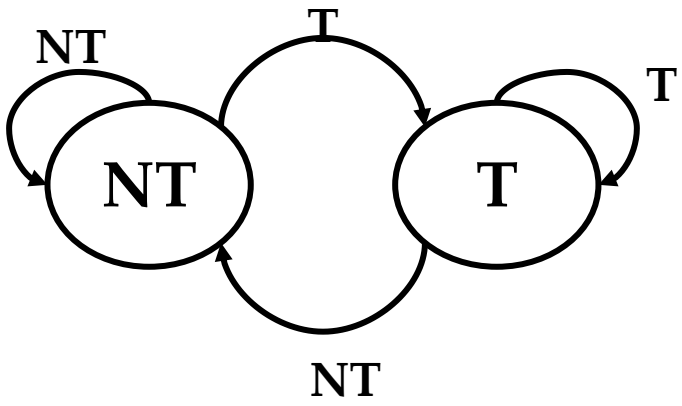
R0 is set to 1
R1 is set to 300

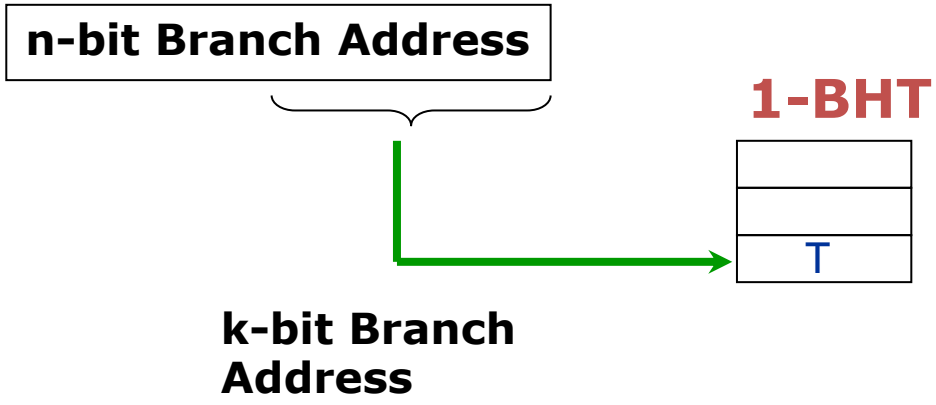


1bit - BHT

R0 is set to 1
R1 is set to 300

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```



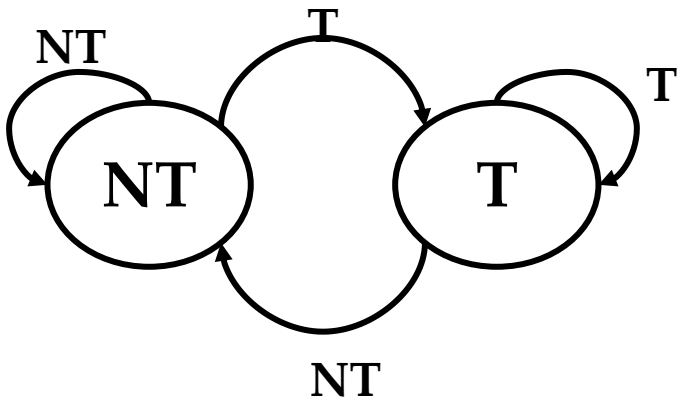


k-bit Branch Address:
Collide
Not collide

1bit - BHT - Not Collide

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

R0 is set to 1
R1 is set to 300



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

2bit - BHT

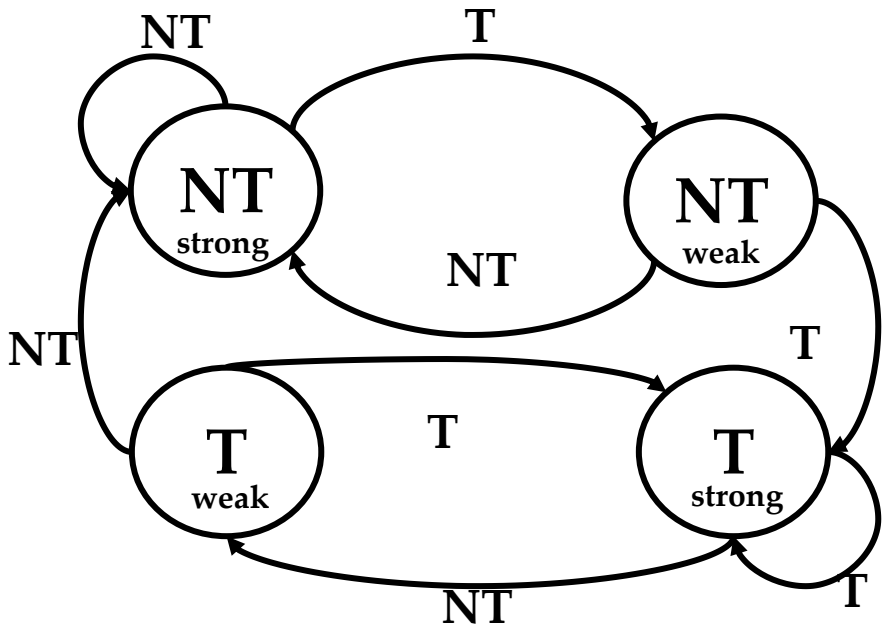
```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

R0 is set to 1
R1 is set to 300

2bit - BHT

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

R0 is set to 1
R1 is set to 300

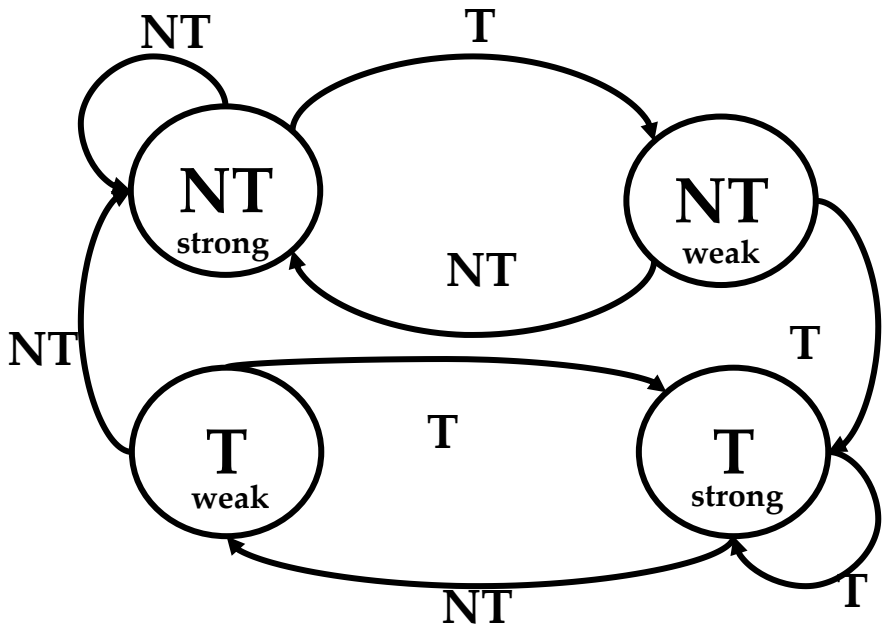


2bit - BHT

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

Let us consider that the branch addresses do NOT collide

R0 is set to 1
R1 is set to 300



2-BHT

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

2-BHT

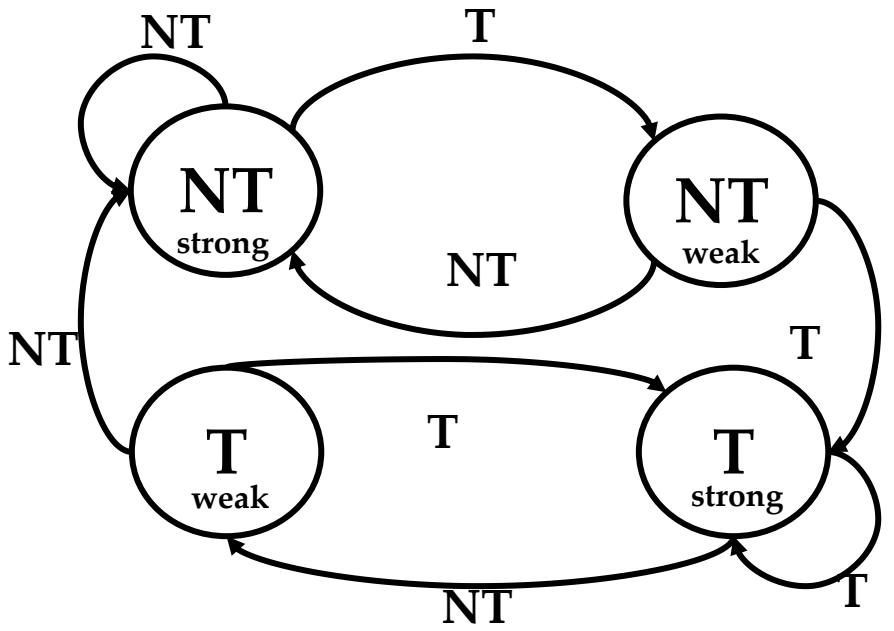
LOOP:	NT_{weak}
LOOP2:	NT_{weak}

2bit - BHT

```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

Let us consider that the branch addresses do NOT collide

R0 is set to 1
R1 is set to 300



2-BHT

LOOP:	T_{weak}
LOOP2:	T_{weak}

2-BHT

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

SUMMARY

Assumption: NO collision

WORST CASES

BEST CASES

Thanks for your attention

Davide Conficconi <davide.conficconi@polimi.it>

Acknowledgements

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

Part of this material comes from:

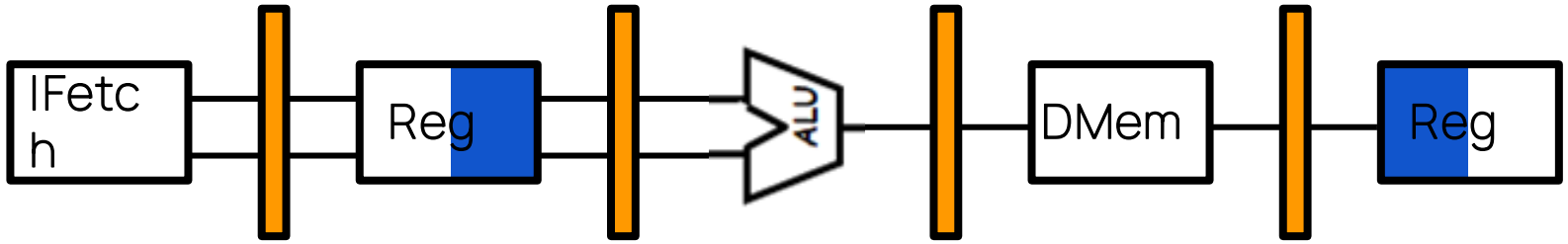
- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- “Digital Design and Computer Architecture” Harris and Harris
- Elsevier Inc. online materials
- Papers/news cited in this lecture

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
```