POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

POLITECNICO MILANO 1863 NECST laboratory

Exercise Session 8

Dynamic Branch Prediction, VLIW, Scoreboard+Cache Misses(Extra:Simple Scheduling)

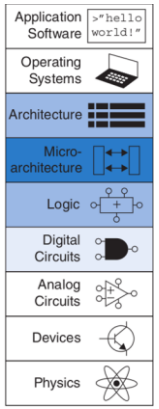
Advanced Computer Architectures

14th May 2025

Davide Conficconi ‹davide.conficconi@polimi.it›

POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

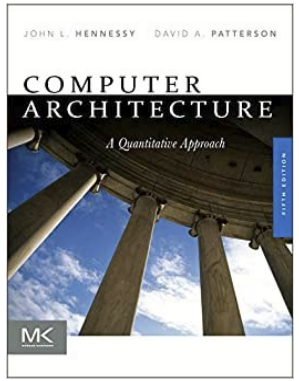
Recall: Material (EVERYTHING OPTIONAL)



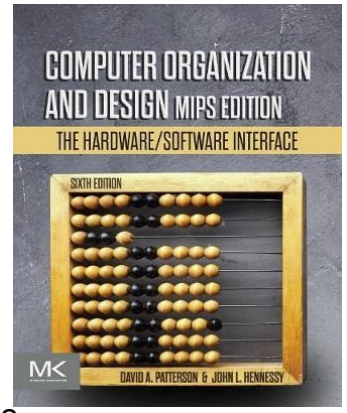
https://webeep.polimi.it/course/view.php?id=14754

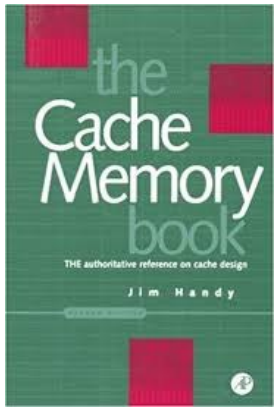
https://tinyurl.com/aca-grid25

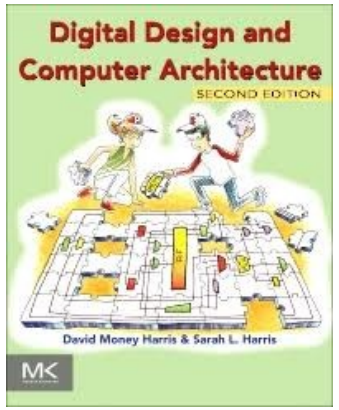
Textbook: Hennessy and Patterson, Computer Architecture: A Quantitative Approach

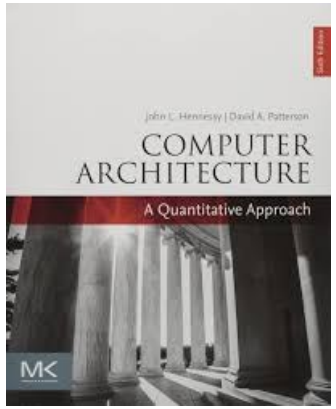


Other Interesting Reference





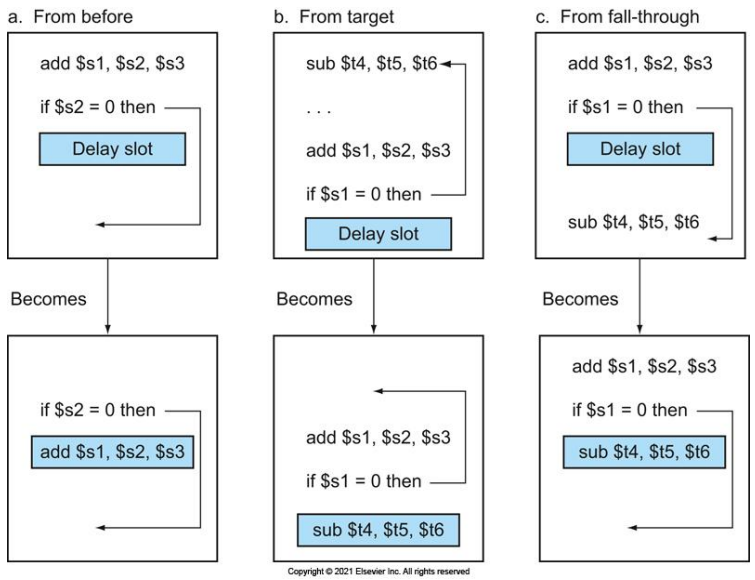


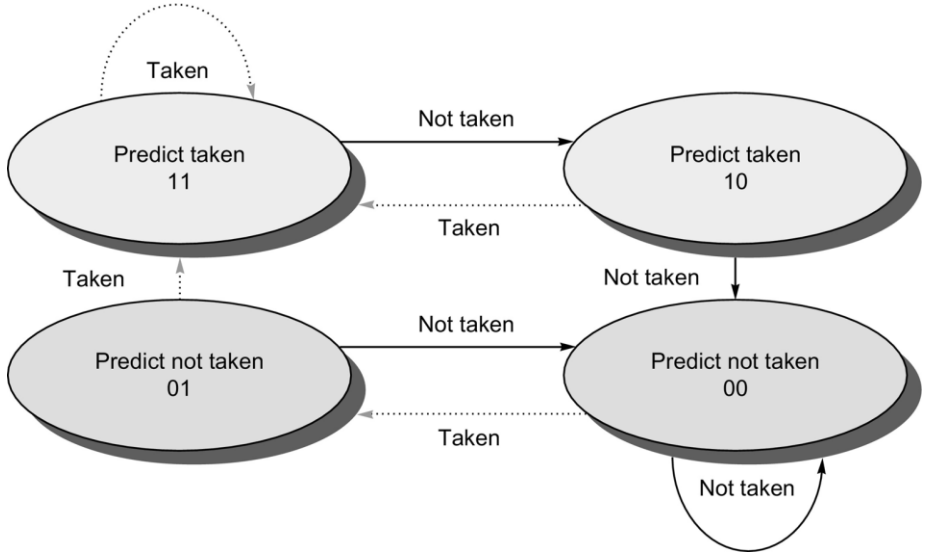


Prediction

[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?

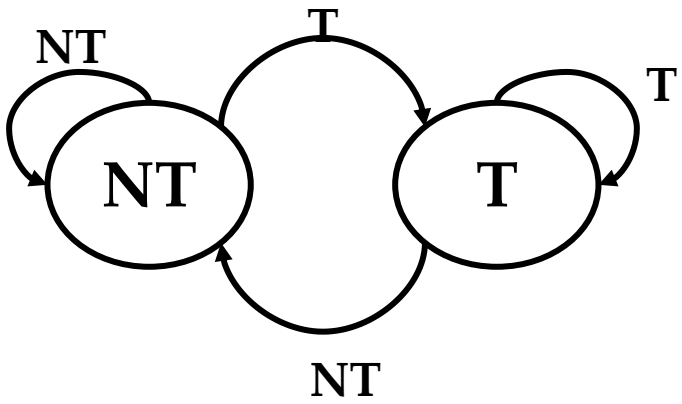
R0 is set to 1
R1 is set to 300

```
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
```

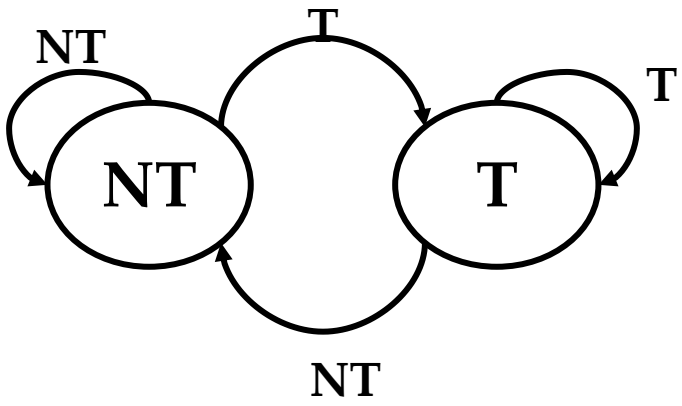
R0 is set to 1
R1 is set to 300

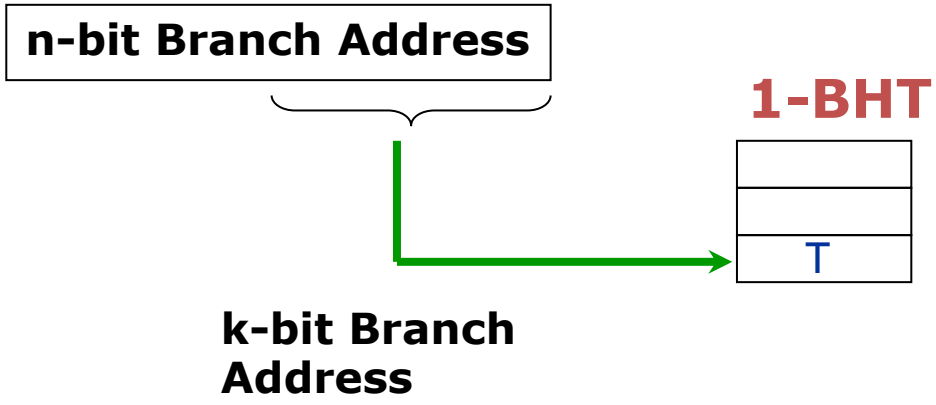


1bit - BHT

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
```



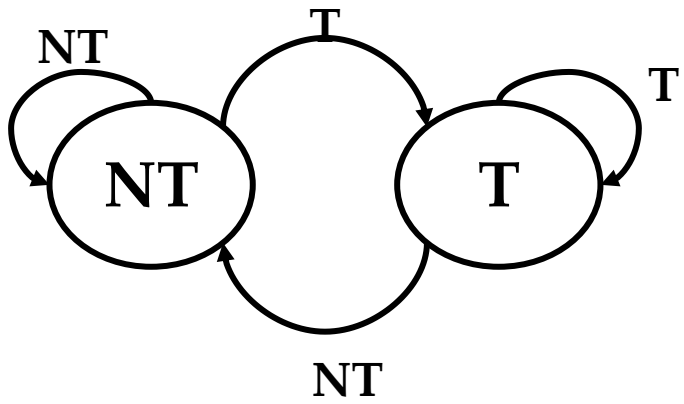


k-bit Branch Address:
Collide
Not collide

1bit - BHT - Not Collide

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
```



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

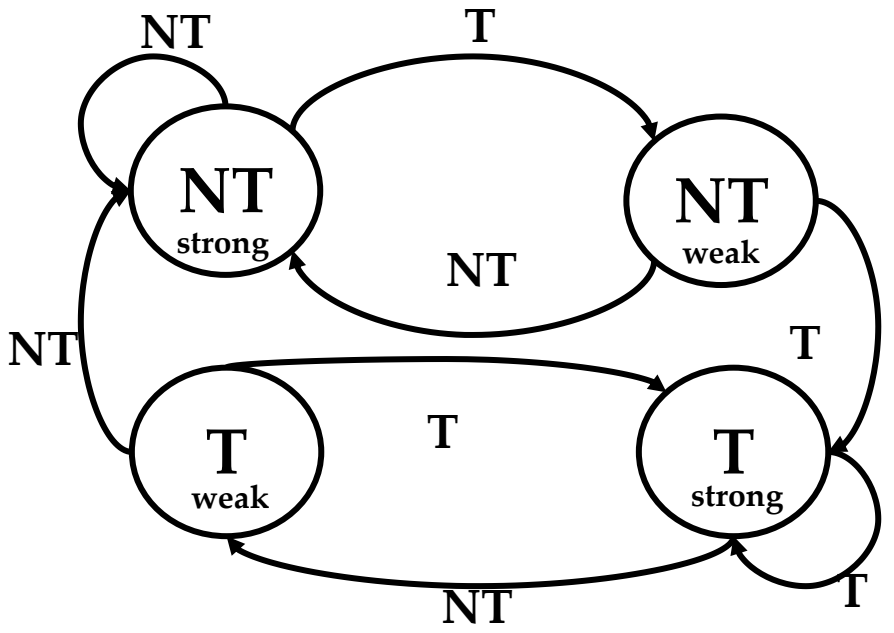
```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

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

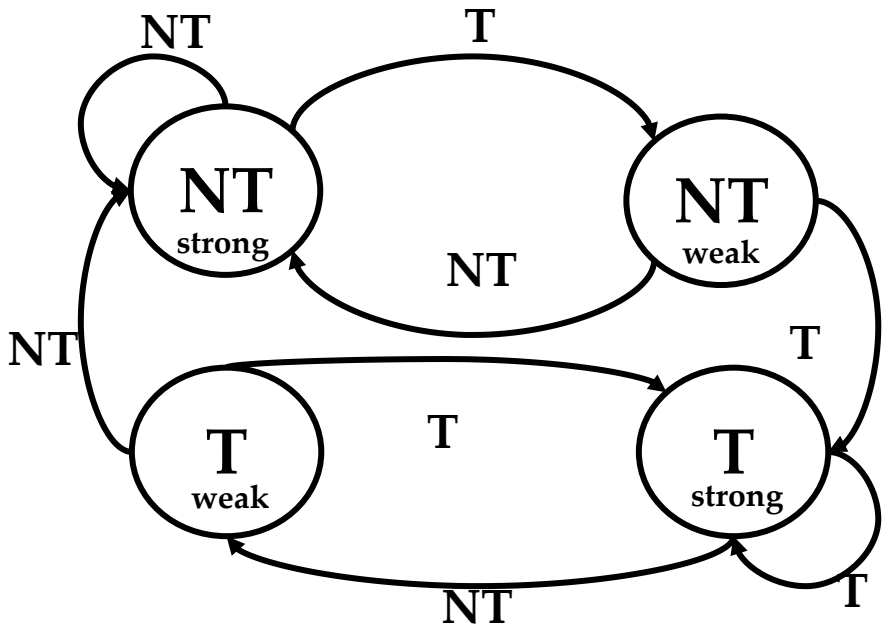


2bit - BHT

```
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

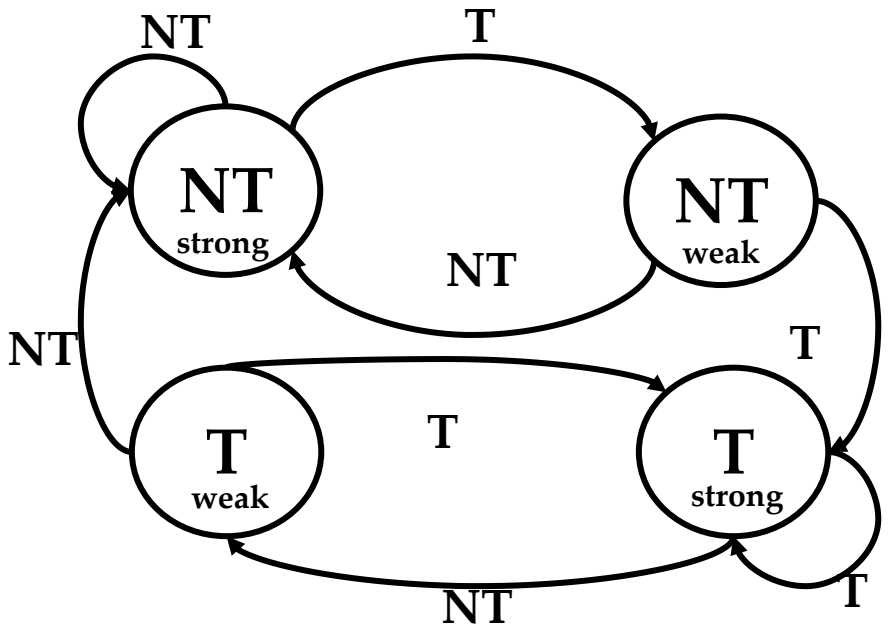
LOOP:	NT_{weak}
LOOP2:	NT_{weak}

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

Exe VLIW.2: Architecture

Details about the **4-operations** VLIW machine with 4 fully **pipelined multi-cycle** functional units:

- 1 **Integer ALU** with **1 cycle latency**
- 1 **Memory Unit** with **3 cycles latency**
- 1 **Floating Point Unit** SUM with **3 cycles latency**
- 1 **FPU** MUL with **5 cycles latency**

What if in-order constraint removed?

- Schedule one iteration of the loop on the 4-opsVLIW machine (we don't write the NOPs) with an **ASAP** and **pipelined** FUs.

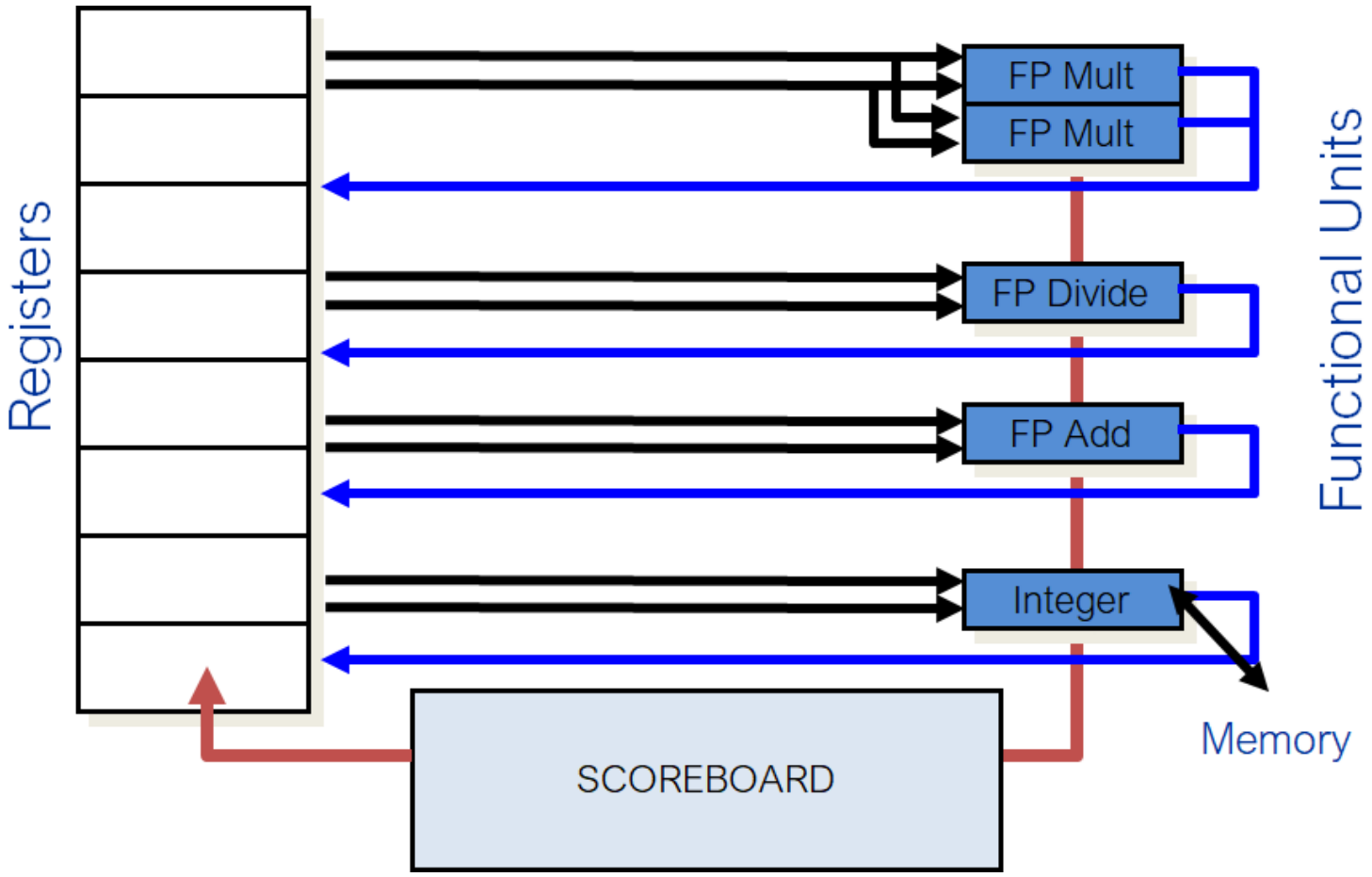
- **Do not** use neither software pipelining nor loop unrolling nor modifying loop indexes

Exe VLIW.2: schedu

```
Loop: LD F0, 0(R1)
      LD F2, 4(R1)
      LD F4, 8(R1)
      LD F6, 12(R1)
      FADD F10,F0,F8
      FADD F12,F2,F8
      FADD F14,F4,F8
      FADD F16,F6,F8
      FMULT F0,F0,F8
      FMULT F2,F2,F8
      FMULT F4,F4,F8
      FMULT F6,F6,F8
      FADD F10,F10,F0
      FADD F12,F12,F2
      FADD F14,F14,F4
      FADD F16,F16,F6
      SD F10, 0(R1)
      SD F12, 4(R1)
      SD F14, 8(R1)
      SD F16, 12(R1)
      ADDI R1,R1,16
      BNE R1 R2 Loop
```

	Integer ALU (1 cc)	Memory Unit (3 cc)	Memory Unit (3 cc)	FPU+ (3 cc)	FPUx (5cc)
C1					
C2					
C3					
C4					
C5					
C6					
C7					
C8					
C9					
C10					
C11					
C12					
C13					
C14					
C15					
C16					
C17					
C18					
C19					
C20					

Exe Scoreboard



Parallel operation in the control data 6600

Recall: 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; TBC WAR not violated	Notify Scoreboard on completion;	WAR &Struct check (FUs will hold results);

Recall: Detailed Scoreboard Pipeline Control
(a.k.a. Instruction Status)

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU) ← op; Fi(FU) ← `D'; Fj(FU) ← `S1'; Fk(FU) ← `S2'; Qj ← Result('S1'); Qk ← Result(`S2'); Rj ← not Qj; Rk ← not Qk; Result('D') ← FU;
Read operands	Rj and Rk	Rj ← No; Rk ← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀ f(if Qj(f)=FU then Rj(f) ← Yes); ∀ f(if Qk(f)=FU then Rk(f) ← Yes); Result(Fi(FU)) ← 0; Busy(FU) ← No

Exe .1 Scoreboard: Code

```
I1:  lw $f1, 0($r0)
I2:  faddi $f1, $f1, C1
I3:  faddi $f2, $f1, C2
I4:  sw $f2, 0($r0)
I5:  lw $f2, 4($r0)
I6:  fadd $f2, $f2, $f2
I7:  sw $f2, 4($r0)
```

Exe .1 Scoreboard: Conflicts

```
I1:  lw $f1, 0($r0)
I2:  faddi $f1, $f1, C1
I3:  faddi $f2, $f1, C2
I4:  sw $f2, 0($r0)
I5:  lw $f2, 4($r0)
I6:  fadd $f2, $f2, $f2
I7:  sw $f2, 4($r0)
```

Exe .1 Scoreboard: Architecture & Cache Miss

3 FPU, latency 2 cc && 4 LDU, latency 4 cc

```
I1:  lw $f1, 0($r0)
I2:  faddi $f1, $f1, C1
I3:  faddi $f2, $f1, C2
I4:  sw $f2, 0($r0)
I5:  lw $f2, 4($r0)
I6:  fadd $f2, $f2, $f2
I7:  sw $f2, 4($r0)
```

I1 cache miss

→ Penalty of 2 cc

I5 cache miss

→ Penalty 5 cc

Exe Scoreboard

3 FPU, latency 2 cc
4 LDU, latency 4 cc

Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Unit
I1:`lw $f1, 0($r0)`					
I2:`faddi $f1, $f1, C1`					
I3:`faddi $f2, $f1, C2`					
I4:`sw $f2, 0($r0)`					
I5:`lw $f2, 4($r0)`					
I6:`fadd $f2, $f2, $f2`					
I7:`sw $f2, 4($r0)`					

I1 cache miss

→ Penalty of 2 cc

I5 cache miss

→ Penalty 5 cc

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:

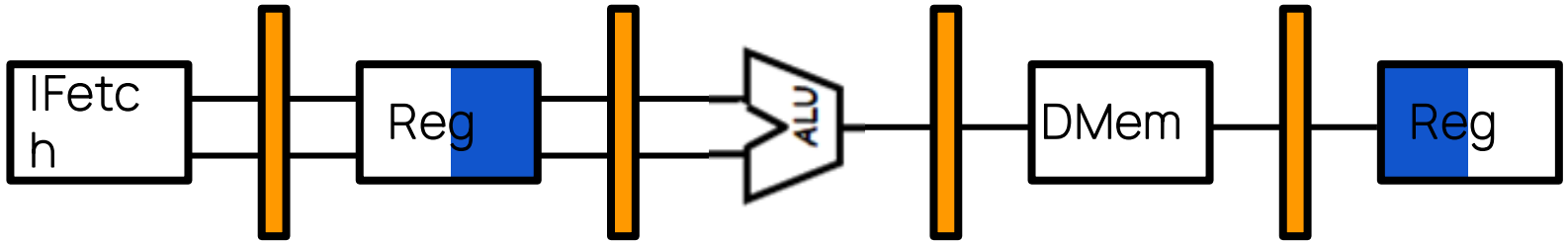
- Spinncloud
- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- “Digital Design and Computer Architecture” Harris and Harris
- Elsevier Inc. online materials
- Papers/news cited in this lecture

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
```