13. Pipelining: Data Hazards

EECS 370 – Introduction to Computer Organization – Winter 2009

Prof. Todd Austin & Prof. Marios Papaefthymiou

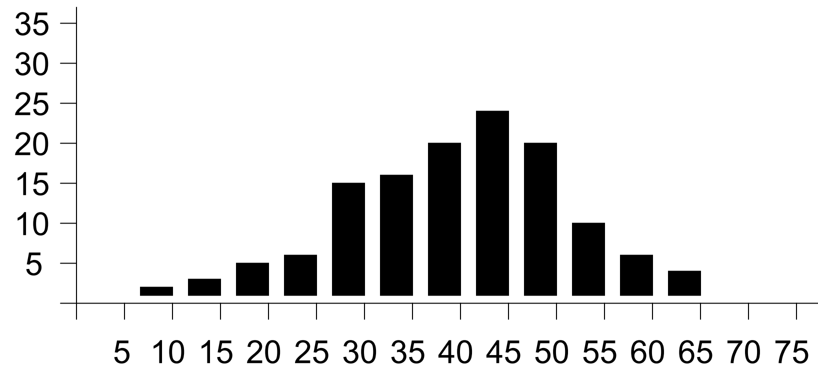
EECS Department
University of Michigan, Ann Arbor, USA

© Austin & Papaefthymiou, 2009
The material in this presentation cannot be copied in any form without our written permission

Announcements

- Programming Contest – Entries accepted until Fri 3/6
- Homework 4 – Due Tue 3/17
- Exam 1 graded – To be handed back at the end of lecture
- Exam 1 regrade requests: Hand in a written request with one paragraph per question to be regraded. We reserve the right to regrade the entire exam.
- Exam 1 regrades are due by Thu 3/12 (one week from today).

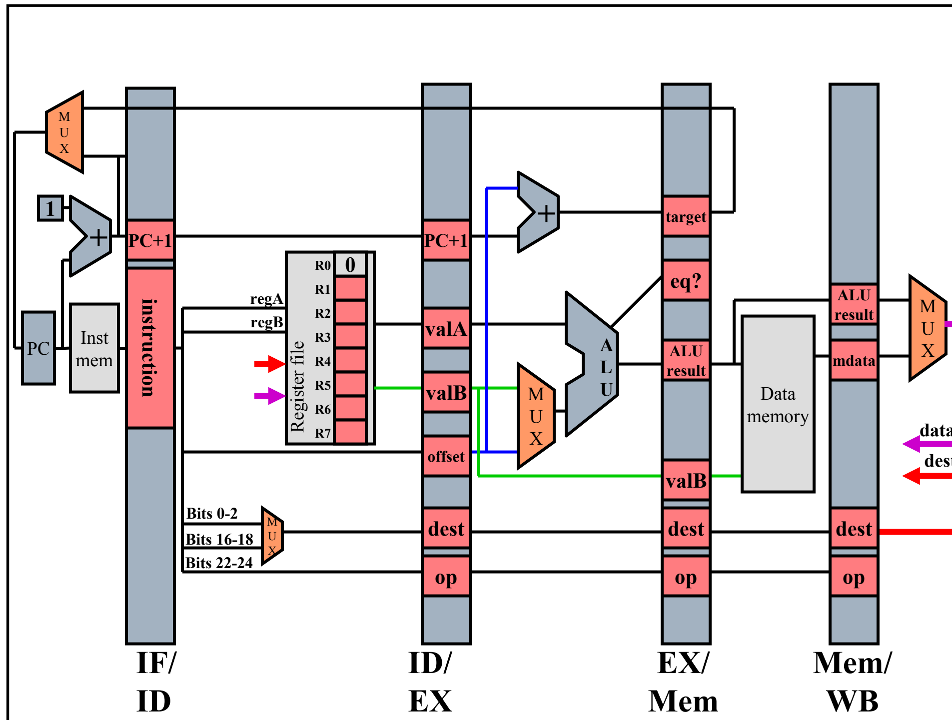
Midterm 1 Statistics



Median = 39   Std. Dev. = 11.5

Pipelining - What can go wrong?

- Data hazards: since register reads occur in stage 2 and register writes occur in stage 5 it is possible to read the wrong value if is about to be written.
- Control hazards: A branch instruction may change the PC, but not until stage 4. What do we fetch before that?
- Exceptions: How do you handle exceptions in a pipelined processor with 5 instructions in flight?
- Today - Data hazards
  - What are they?
  - How do you detect them?
  - How do you deal with them?

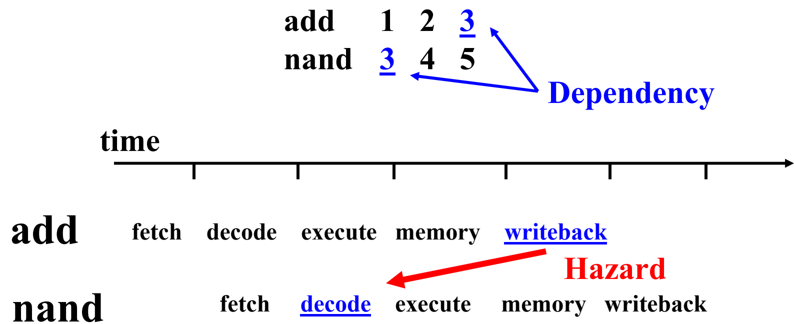


Pipeline function for ADD

- Fetch: read instruction from memory
- Decode: **read source operands from reg**
- Execute: calculate sum
- Memory: pass results to next stage
- Writeback: **write sum into register file**

© Austin & Papaefthymiou – 2009

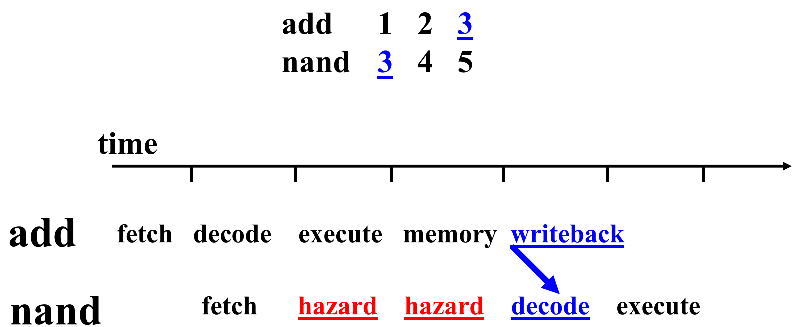
Data Hazards



If not careful, nand will read the wrong value of R3

© Austin & Papaefthymiou – 2009

Data Hazards



Assume Register File gives the right value of R3 when read/written during same cycle. This is consistent with the book and the MIPS architecture, but not Project 3.

© Austin & Papaefthymiou – 2009

Class Problem

Which data hazards do you see?

```
add 1 2 3
nand 3 4 5
add 6 3 7
lw 3 6 10
sw 6 2 12
```


Three approaches to handling data hazards

- Avoid
  - Make sure there are no hazards in the code
- Detect and Stall
  - If hazards exist, stall the processor until they go away.
- Detect and Forward
  - If hazards exist, fix up the pipeline to get the correct value (if possible)

Handling data hazards I: Avoid all hazards

- Assume the programmer (or the compiler) knows about the processor implementation.
  - Make sure no hazards exist.
    - Put noops between any dependent instructions.

```
add  1 2 3   ← write R3 in cycle 5
noop
noop
nand 3 4 5   ← read R3 in cycle 5
```

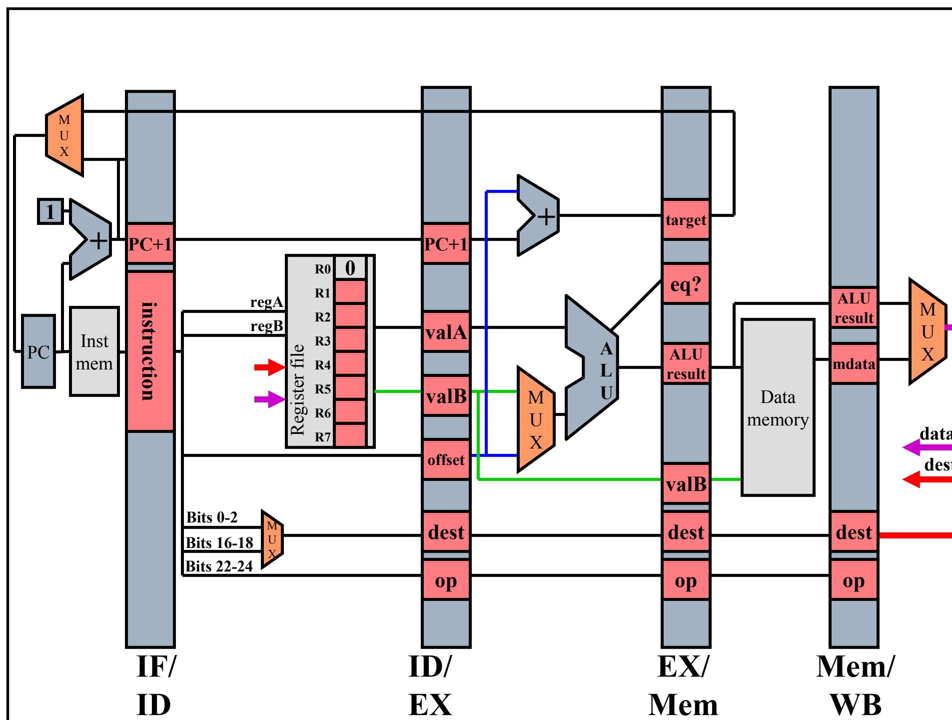

Problems with this solution

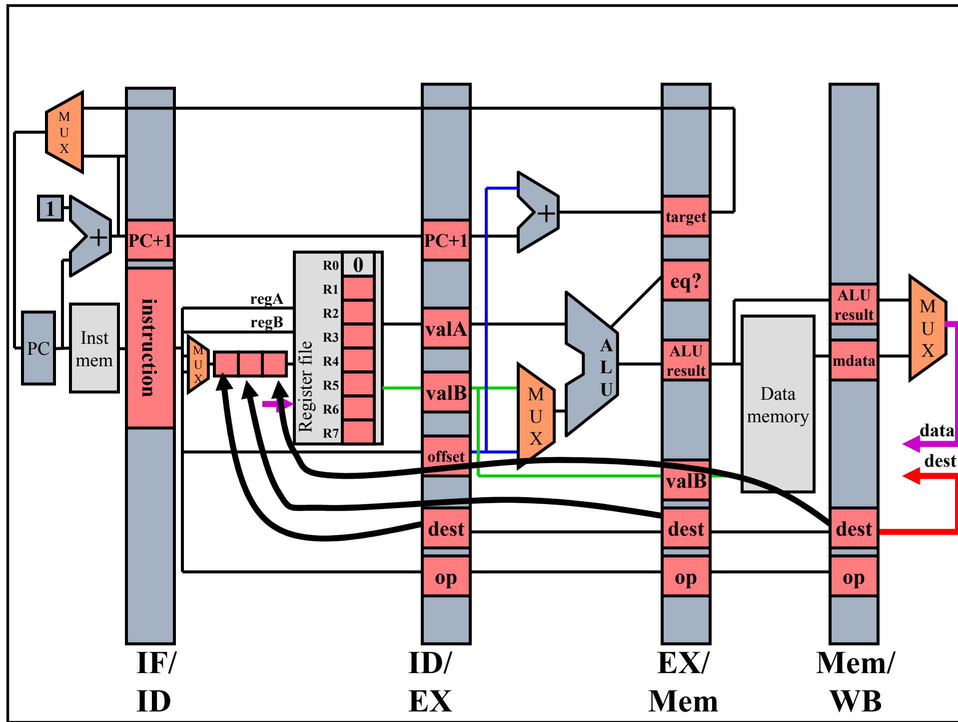
- Old programs (legacy code) may not run correctly on new implementations
  - Longer pipelines need more noops
- Programs get larger as noops are included
  - Especially a problem for machines that try to execute more than one instruction every cycle
  - Intel EPIC: Often 25% - 40% of instructions are noops
- Program execution is slower
  - CPI is 1, but some instructions are noops

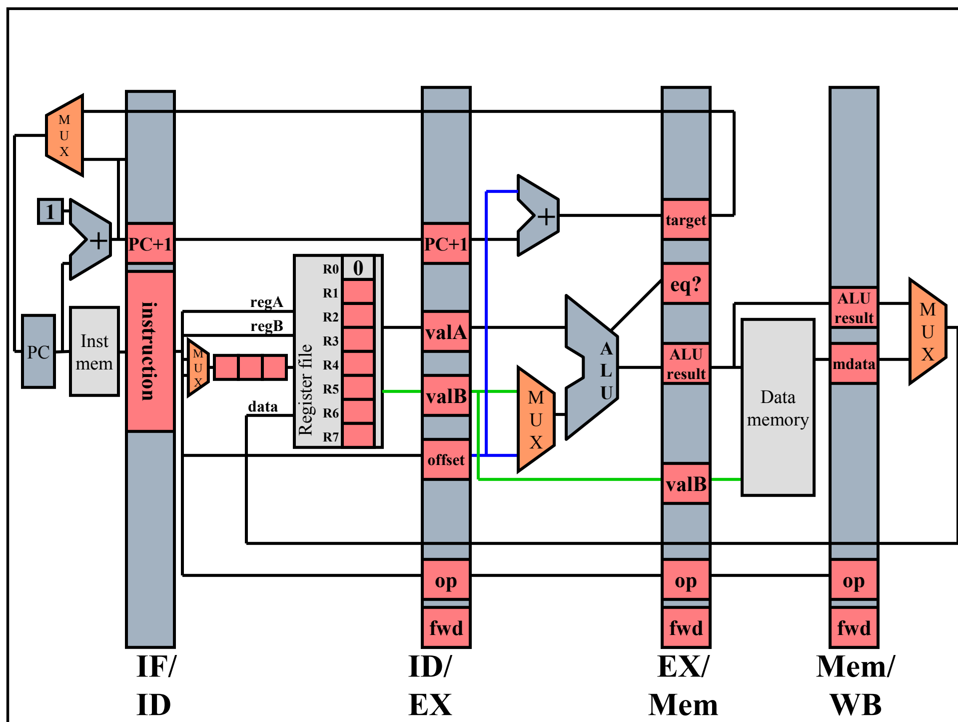
Handling data hazards II: Detect and stall until ready

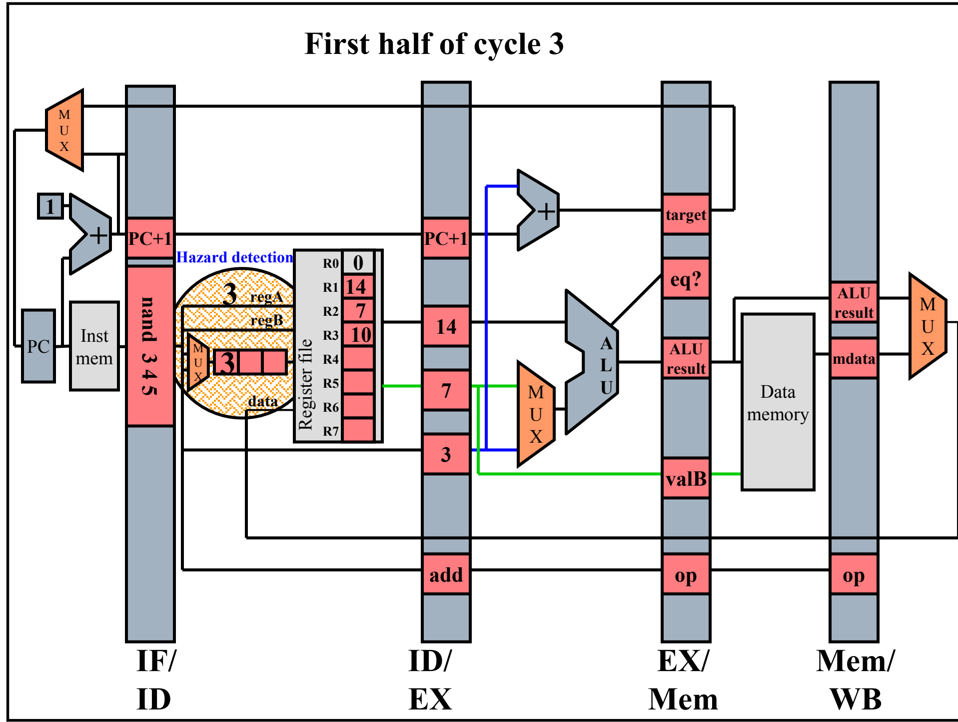
- Detect:
  - Compare regA with previous DestRegs
    - 3 bit operand fields
  - Compare regB with previous DestRegs
    - 3 bit operand fields
- Stall:
  - Keep current instructions in fetch and decode
  - Pass a noop to execute

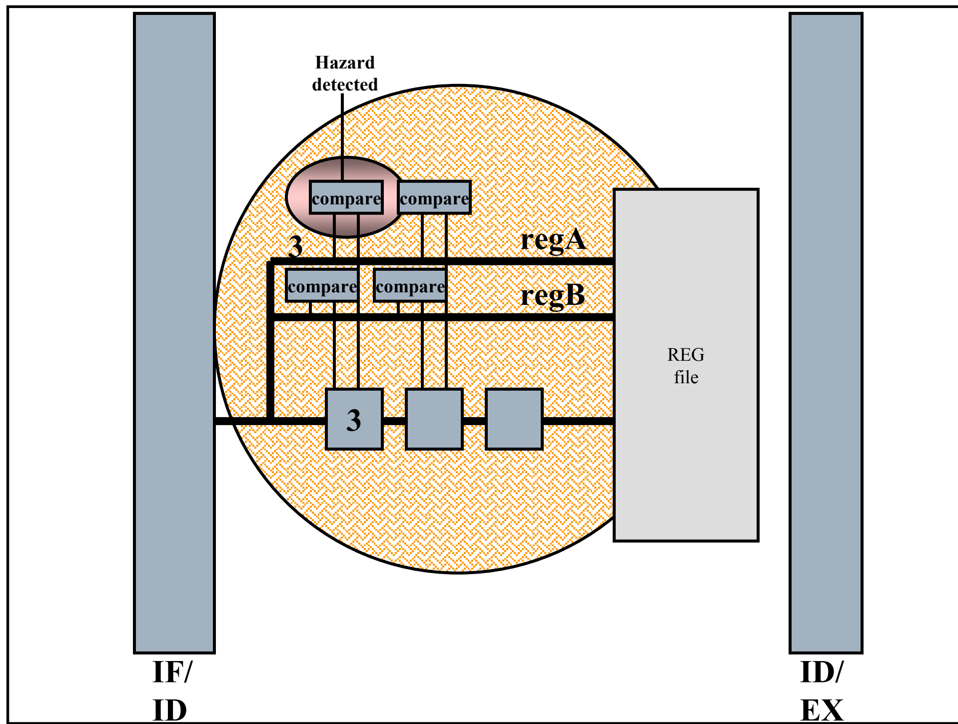
© Austin & Papaefthymiou – 2009

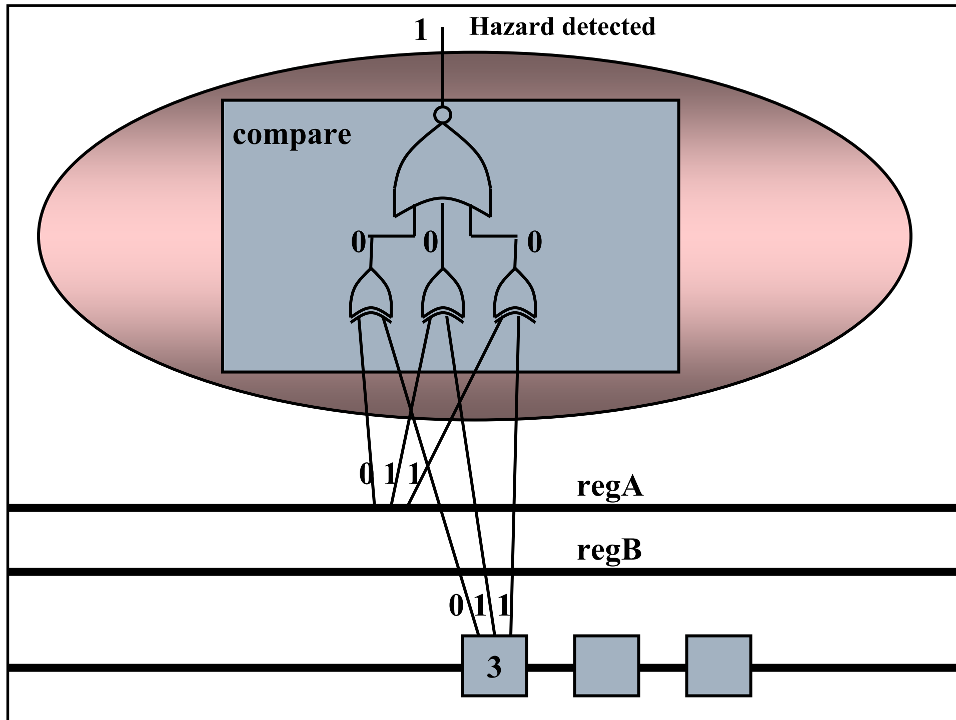



MUX
1
PC+1
instruction
PC
Inst mem
regA
regB
Register file
R0
R1
R2
R3
R4
R5
R6
R7
0
PC+1
valA
valB
offset
dest
op
target
eq?
ALU result
valB
dest
op
ALU
Data memory
ALU result
mdata
dest
op
data
dest
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


MUX
1
PC+1
instruction
PC
Inst mem
regA
regB
data
Register file
R0
R1
R2
R3
R4
R5
R6
R7
0
PC+1
valA
valB
offset
op
fwd
target
eq?
ALU result
valB
op
fwd
ALU
Data memory
ALU result
mdata
op
fwd
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


First half of cycle 3
M U X
1
+
PC
Inst mem
PC+1
nand 3 4 5
Hazard detection
3 regA
regB
M U X
3
data
Register file
R0 0
R1 14
R2 7
R3 10
R4
R5
R6
R7
PC+1
14
7
3
add
+
M U X
A L U
target
eq?
ALU result
valB
op
Data memory
ALU result
mdata
op
M U X
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


Hazard detected
compare
compare
3
regA
compare
compare
regB
REG file
3
IF/ ID
ID/ EX



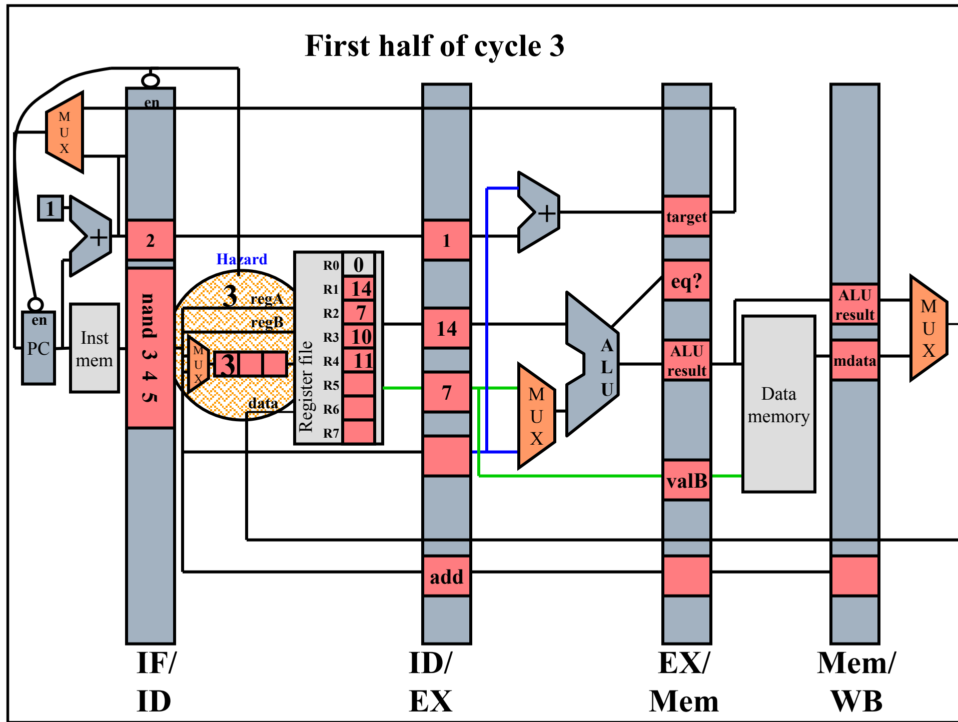
Handling data hazards II: Detect and stall until ready

- Detect:
  - Compare regA with previous DestReg
    - 3 bit operand fields
  - Compare regB with previous DestReg
    - 3 bit operand fields
- Stall:

  Keep current instructions in fetch and decode

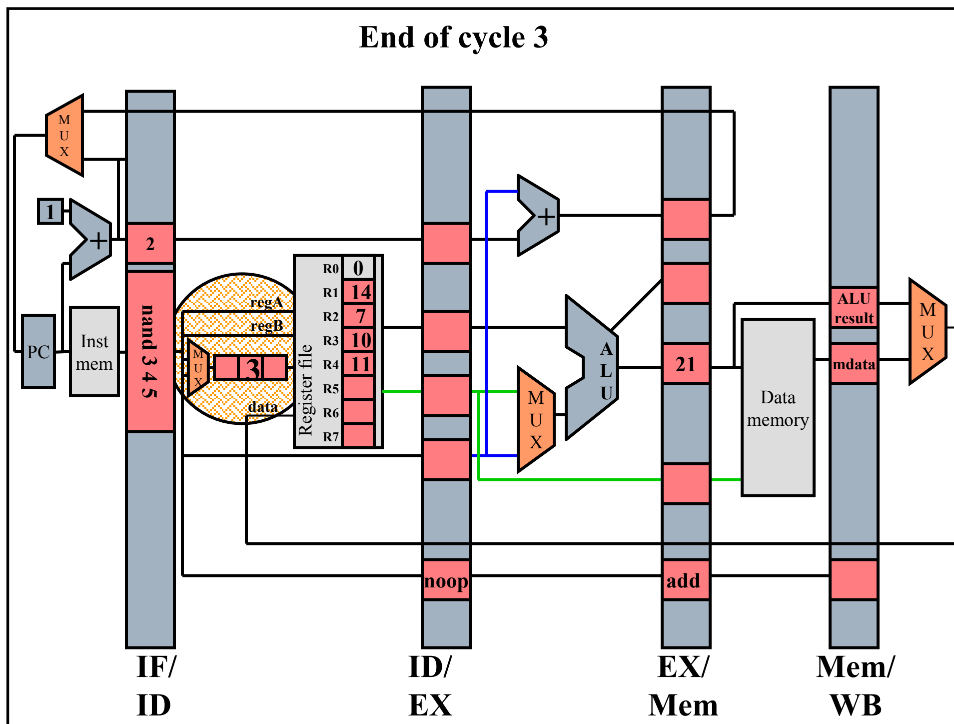
  Pass a noop to execute

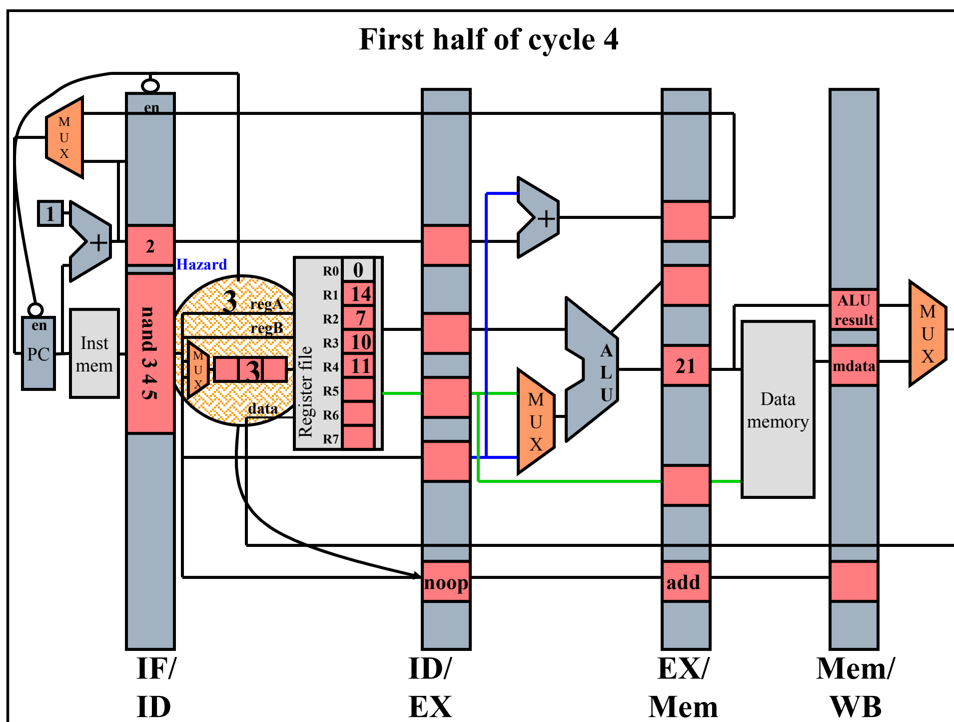
The University of Michigan
© Austin & Papaefthymiou – 2009

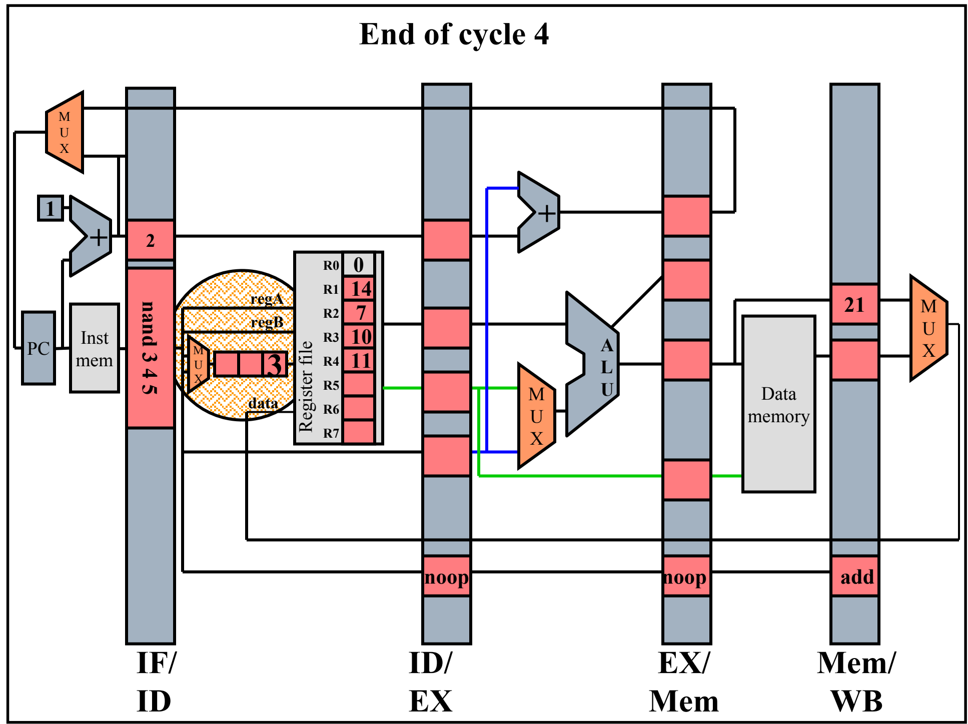


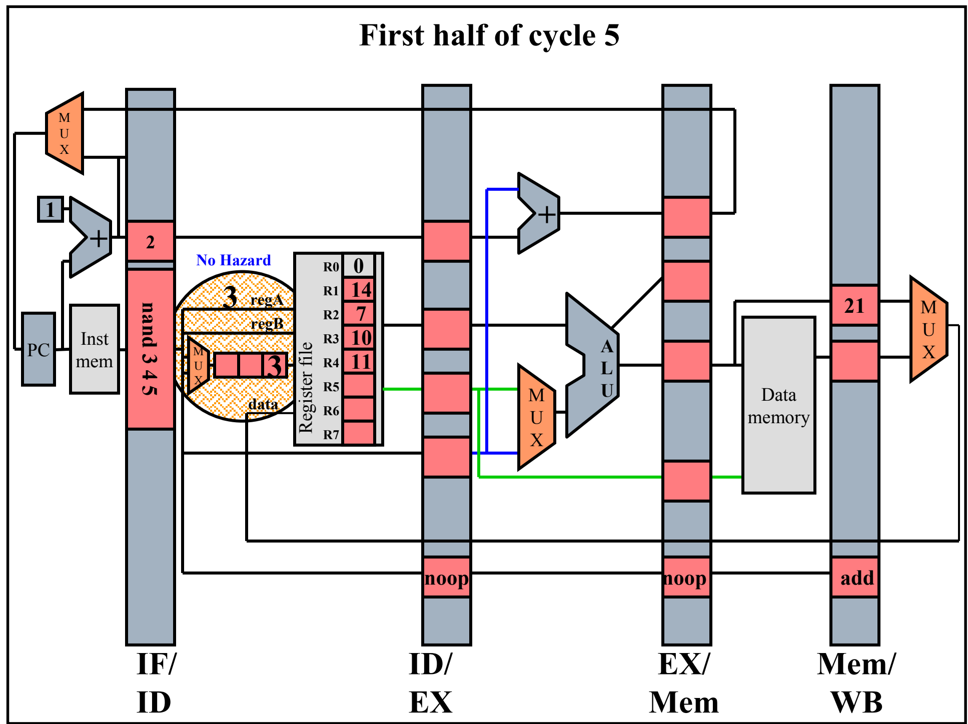
Handling data hazards II:
Detect and stall until ready

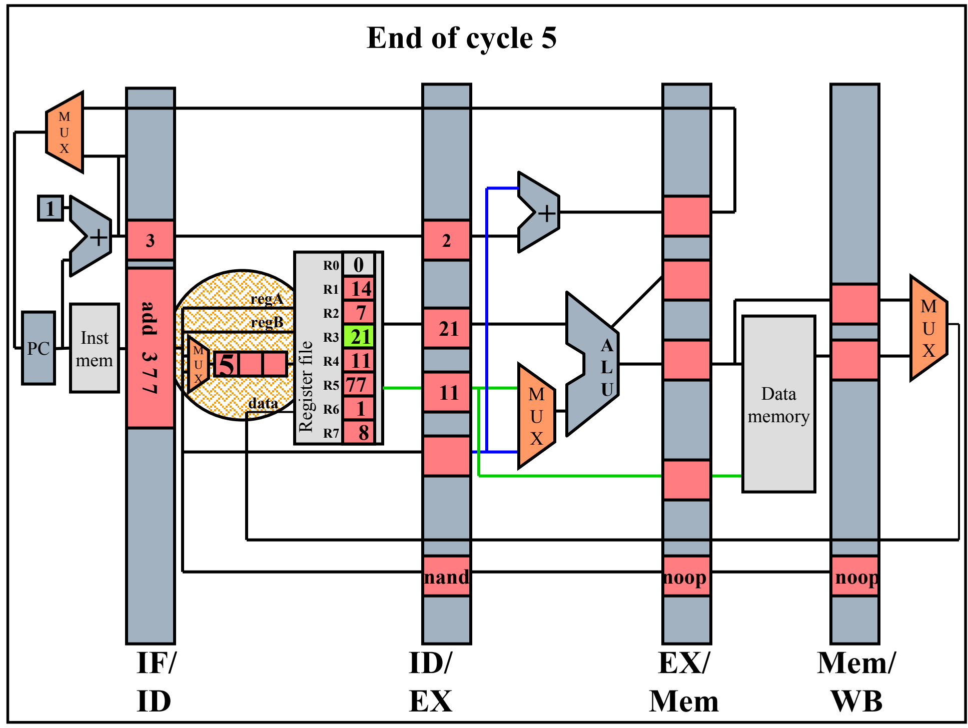
- Detect:
  - Compare regA with previous DestReg
    - 3 bit operand fields
  - Compare regB with previous DestReg
    - 3 bit operand fields
- Stall:
  - Keep current instructions in fetch and decode
  - <u>**Pass a noop to execute**</u>


End of cycle 3
M U X
1
2
nand 3 4 5
regA
regB
3
data
Register file
R0 0
R1 14
R2 7
R3 10
R4 11
R5
R6
R7
PC
Inst mem
noop
A L U
21
add
Data memory
ALU result
mdata
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


First half of cycle 4
en
M U X
1
2
Hazard
nand 3 4 5
3
regA
regB
data
Register file
R0 0
R1 14
R2 7
R3 10
R4 11
R5
R6
R7
en
PC
Inst mem
noop
A L U
21
add
Data memory
ALU result
mdata
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


End of cycle 4
M
U
X
1
+
2
PC
Inst
mem
nand 3 4 5
regA
regB
3
data
Register file
R0 0
R1 14
R2 7
R3 10
R4 11
R5
R6
R7
M
U
X
A
L
U
Data
memory
21
M
U
X
noop
noop
add
IF/
ID
ID/
EX
EX/
Mem
Mem/
WB


First half of cycle 5
M
U
X
1
+
2
PC
Inst
mem
nand 3 4 5
No Hazard
3
regA
regB
3
data
Register file
R0 0
R1 14
R2 7
R3 10
R4 11
R5
R6
R7
M
U
X
A
L
U
Data
memory
21
M
U
X
noop
noop
add
IF/
ID
ID/
EX
EX/
Mem
Mem/
WB



Time Graph

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nand 3 4 5		IF	no op	no op	ID	EX	ME	WB					
add 6 3 7					IF	ID	EX	ME	WB				
lw 3 6 10						IF	ID	EX	ME	WB			
sw 6 2 12							IF	no op	no op	ID	EX	ME	WB

© Austin & Papaefthymiou – 2009

Problems with detect and stall

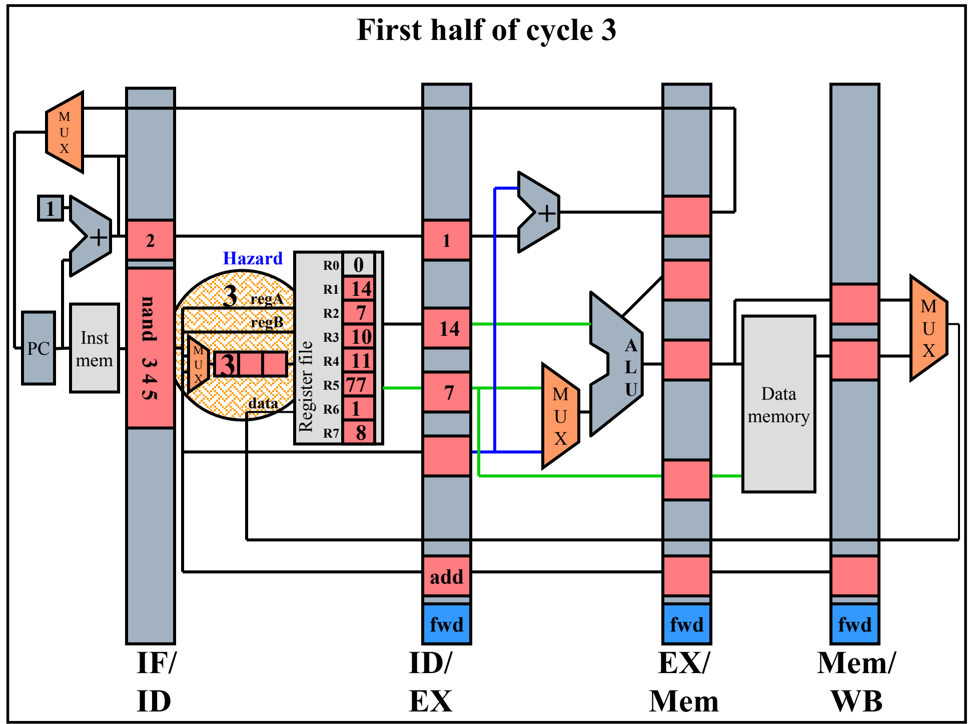
- CPI increases every time a hazard is detected!
- Is that necessary? Not always!
  - Re-route the result of the add to the nand
    - nand no longer needs to read R3 from reg file
    - It can get the data later (when it is ready)
    - This lets us complete the decode this cycle
      - But we need more control to remember that the data that we aren't getting from the reg file at this time will be found elsewhere in the pipeline at a later cycle.

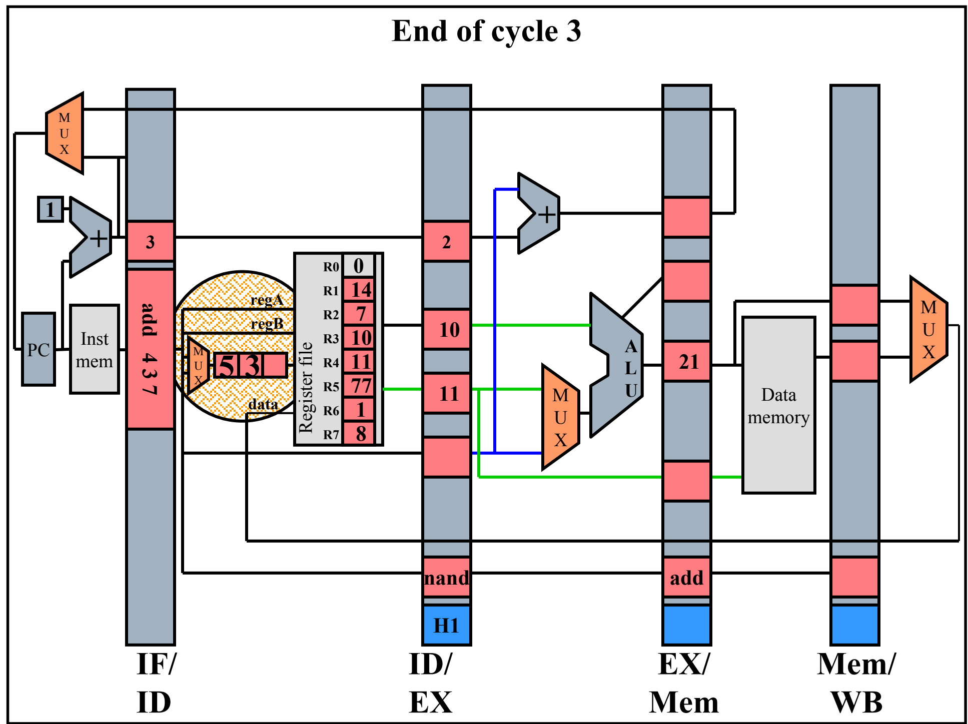
© Austin & Papaefthymiou – 2009

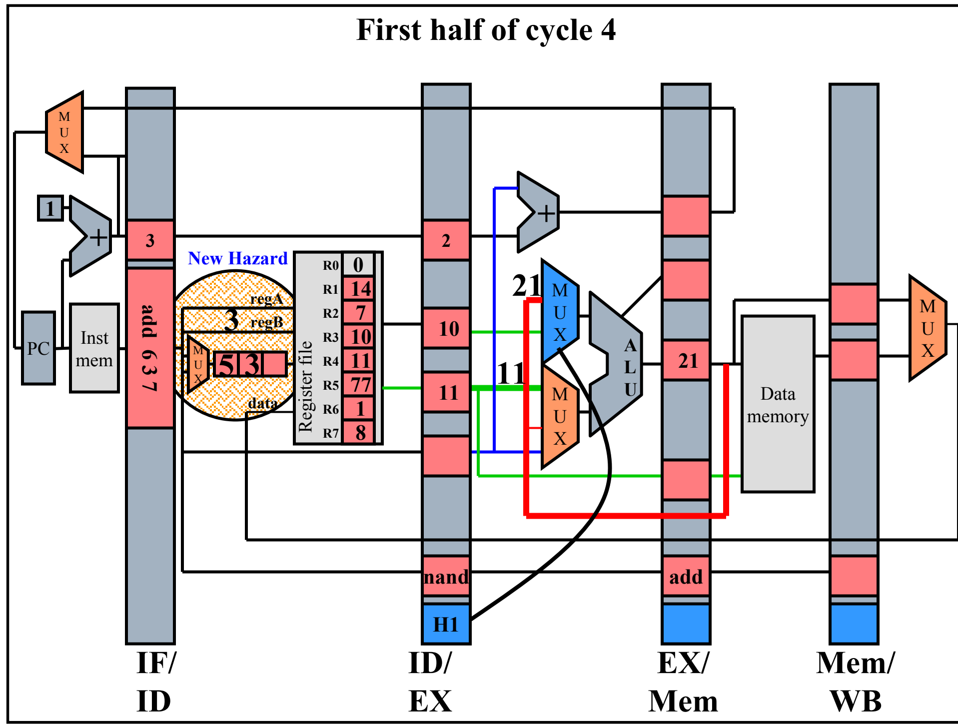
Handling data hazards III: Detect and forward

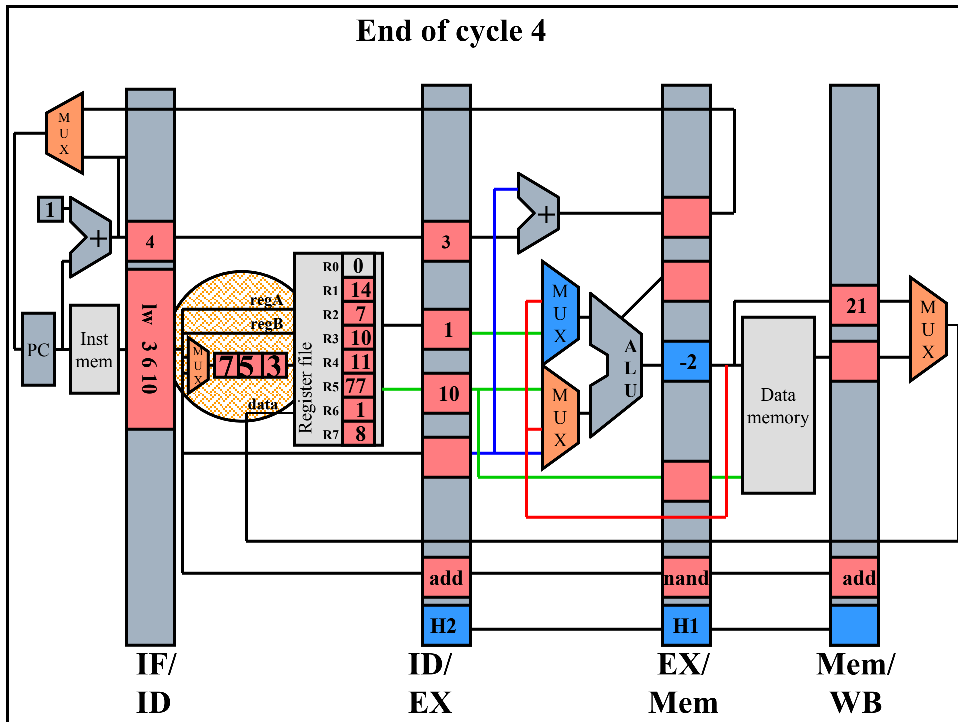
- Detect: same as detect and stall
  - Except that all 4 hazards are treated differently
    - i.e., you can't logical-OR the 4 hazard signals
- Forward:
  - New bypass datapaths route computed data to where it is needed
  - New MUX and control to pick the right data
- Beware: Stalling may still be required even in the presence of forwarding

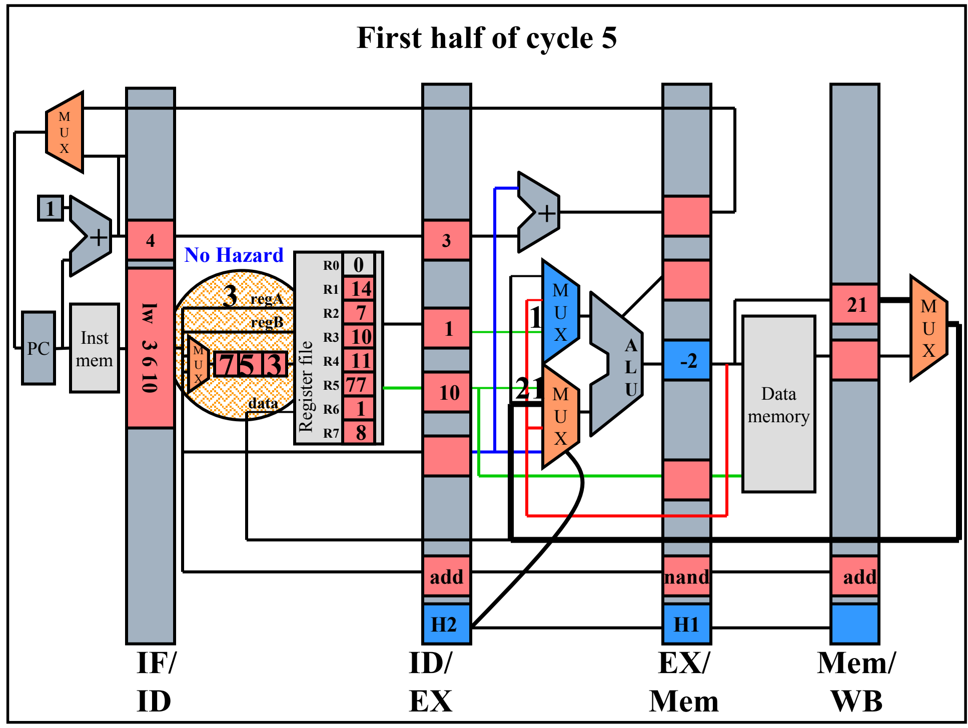
© Austin & Papaefthymiou – 2009

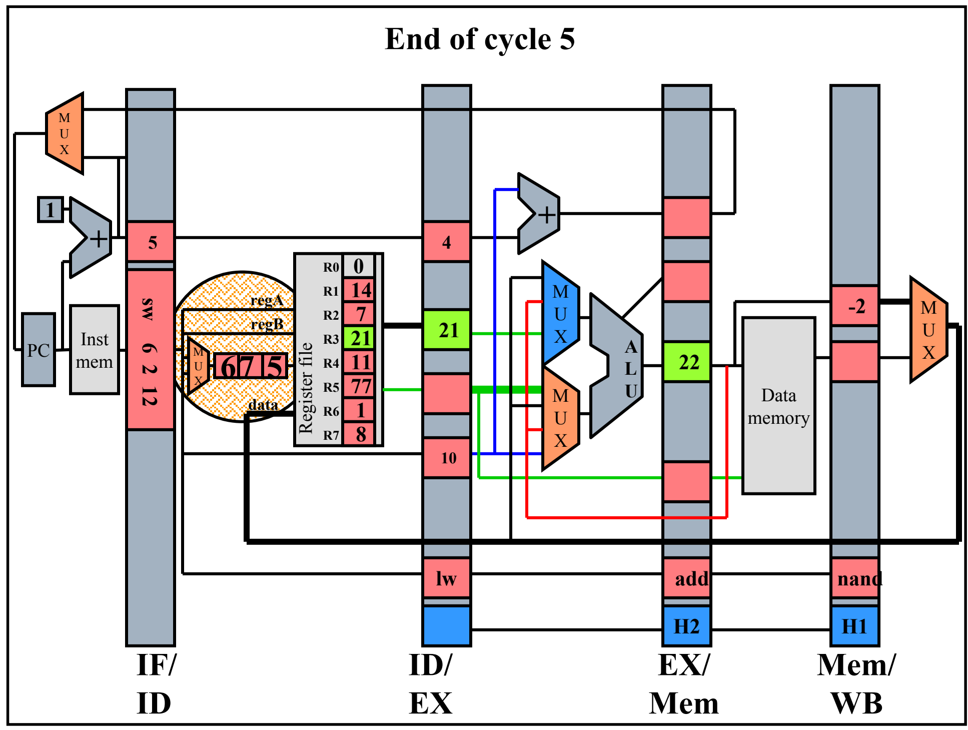

First half of cycle 3
MUX
1
+
2
nand 3 4 5
Hazard
3
regA
regB
MUX
3
data
Register file
R0 0
R1 14
R2 7
R3 10
R4 11
R5 77
R6 1
R7 8
PC
Inst mem
1
14
7
add
fwd
+
MUX
ALU
fwd
Data memory
fwd
MUX
IF/ID
ID/EX
EX/Mem
Mem/WB

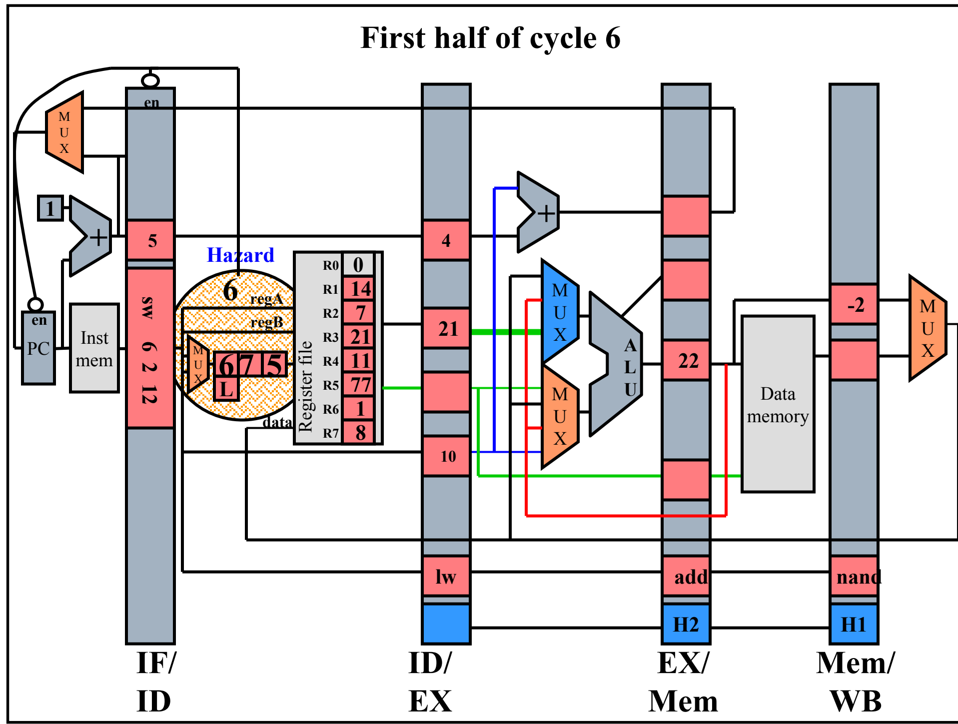

End of cycle 3
MUX
1
+
3
add 4 3 7
regA
regB
MUX
5 3
data
Register file
R0 0
R1 14
R2 7
R3 10
R4 11
R5 77
R6 1
R7 8
PC
Inst mem
2
10
11
nand
H1
+
MUX
ALU
21
add
Data memory
MUX
IF/ID
ID/EX
EX/Mem
Mem/WB

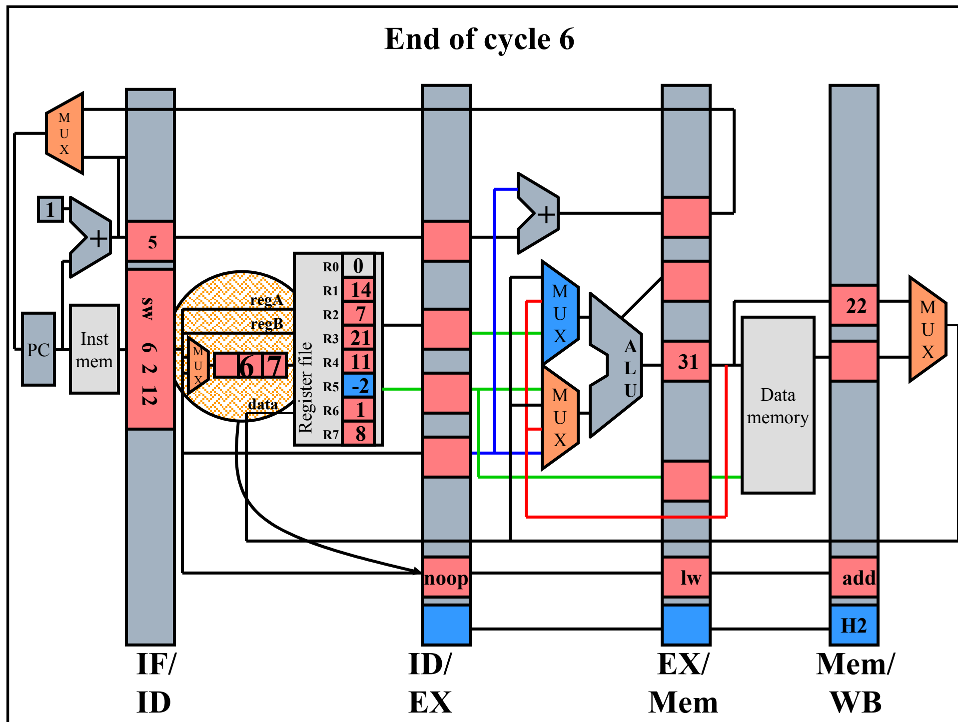

First half of cycle 4
New Hazard
add 6 3 7
Register file
regA
regB
data
3
5 3
R0 0
R1 14
R2 7
R3 10
R4 11
R5 77
R6 1
R7 8
PC
Inst mem
2
10
11
21
11
nand
H1
21
add
ALU
Data memory
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB

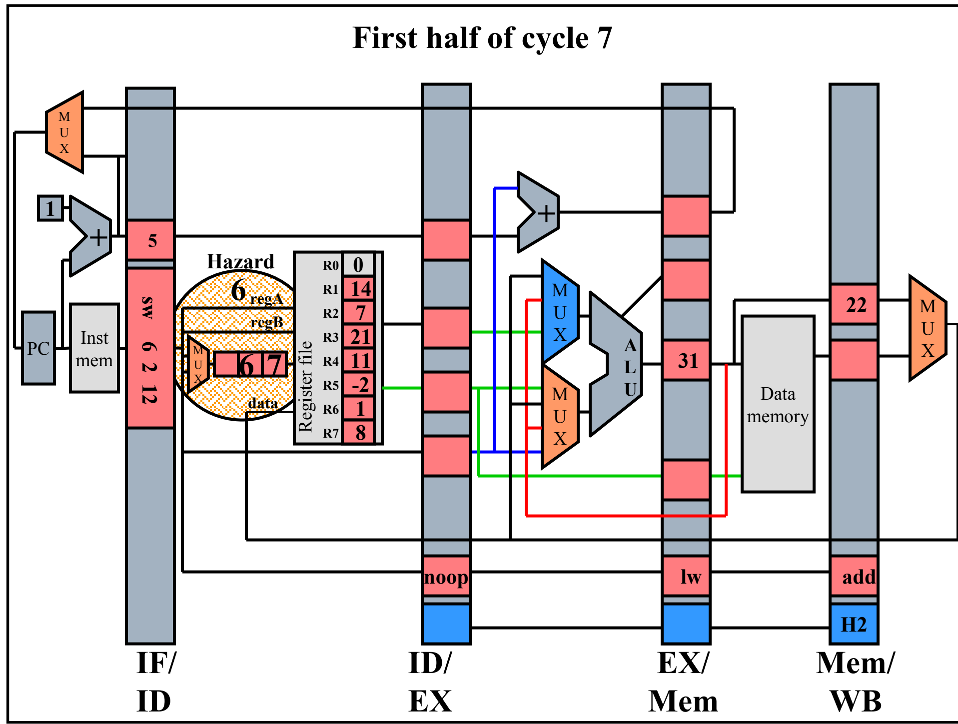

End of cycle 4
lw 3 6 10
Register file
regA
regB
data
7 5 3
R0 0
R1 14
R2 7
R3 10
R4 11
R5 77
R6 1
R7 8
PC
Inst mem
4
3
1
10
add
H2
-2
nand
H1
21
add
ALU
Data memory
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB

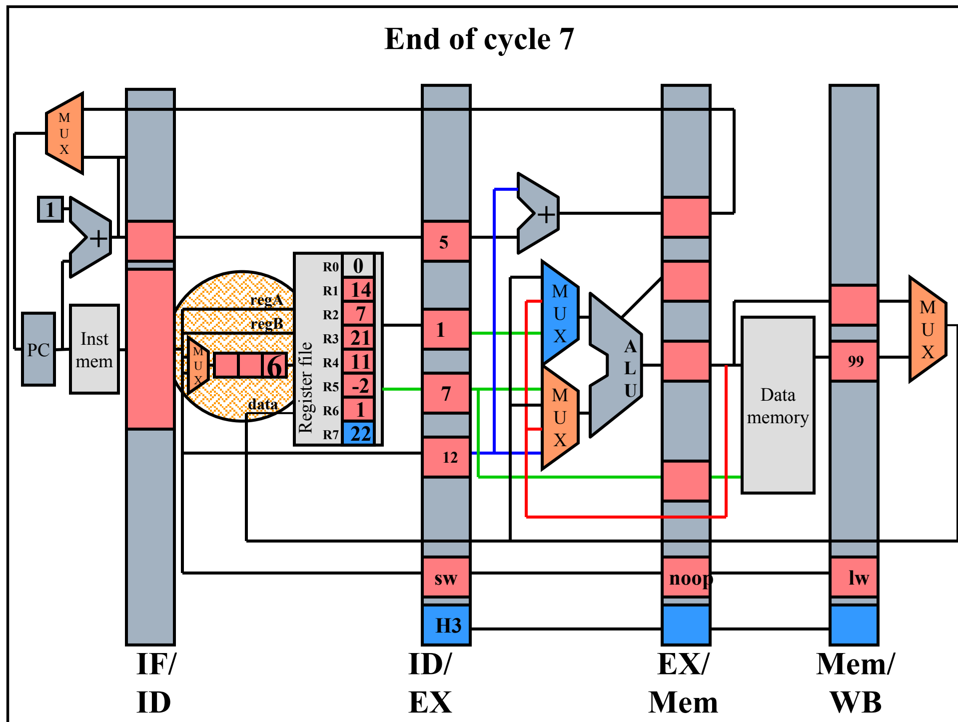

First half of cycle 5
No Hazard
lw 3 6 10
Inst mem
PC
Register file
regA
regB
data
Data memory
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB
add
nand
add
H2
H1

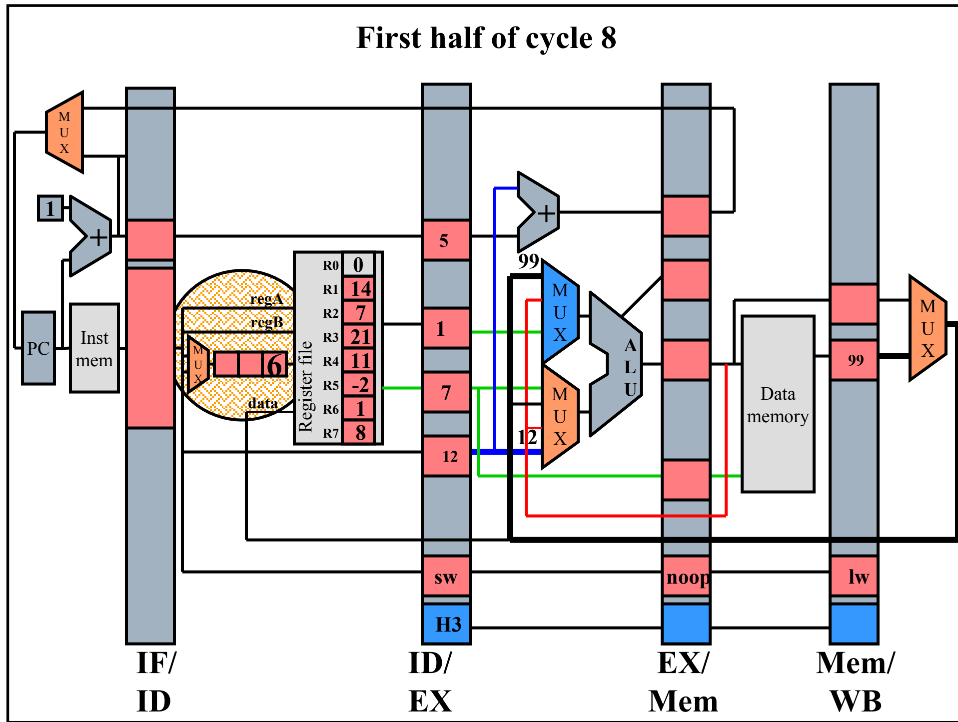

End of cycle 5
sw 6 2 12
Inst mem
PC
Register file
regA
regB
data
Data memory
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB
lw
add
nand
H2
H1

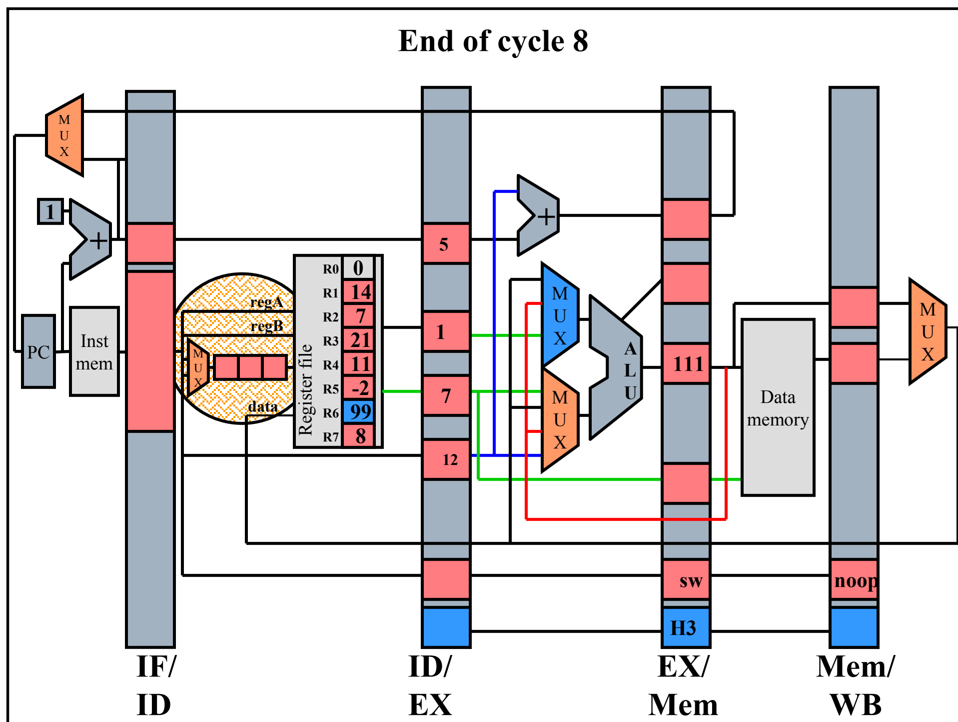

First half of cycle 6
M
U
X
en
en
PC
1
+
Inst
mem
5
sw 6 2 12
Hazard
6
regA
regB
M
U
X
6 7 5
L
data
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 77
R6 1
R7 8
4
21
10
lw
+
M
U
X
M
U
X
A
L
U
22
add
H2
Data
memory
-2
nand
H1
M
U
X
IF/
ID
ID/
EX
EX/
Mem
Mem/
WB


End of cycle 6
M
U
X
PC
1
+
Inst
mem
5
sw 6 2 12
regA
regB
M
U
X
6 7
data
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 -2
R6 1
R7 8
noop
+
M
U
X
M
U
X
A
L
U
31
lw
Data
memory
22
add
H2
M
U
X
IF/
ID
ID/
EX
EX/
Mem
Mem/
WB


First half of cycle 7
MUX
1
PC
Inst mem
5
sw 6 2 12
Hazard
6
regA
regB
6 7
data
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 -2
R6 1
R7 8
MUX
MUX
ALU
31
Data memory
22
MUX
noop
lw
add
H2
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


End of cycle 7
MUX
1
PC
Inst mem
regA
regB
6
data
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 -2
R6 1
R7 22
5
1
7
12
MUX
MUX
ALU
Data memory
99
MUX
sw
H3
noop
lw
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


First half of cycle 8
MUX
1
PC
Inst mem
regA
regB
data
6
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 -2
R6 1
R7 8
5
1
7
12
sw
H3
99
12
MUX
ALU
noop
Data memory
99
lw
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB


End of cycle 8
MUX
1
PC
Inst mem
regA
regB
data
Register file
R0 0
R1 14
R2 7
R3 21
R4 11
R5 -2
R6 99
R7 8
5
1
7
12
MUX
ALU
111
sw
H3
Data memory
noop
IF/ ID
ID/ EX
EX/ Mem
Mem/ WB

Time Graph

Time:	1	2	3	4	5	6	7	8	9	10	11	12	13
add 1 2 3	IF	ID	EX	ME	WB								
nand 3 4 5		IF	ID	EX	ME	WB							
add 6 3 7			IF	ID	EX	ME	WB						
lw 3 6 10				IF	ID	EX	ME	WB					
sw 6 2 12					IF	no op	ID	EX	ME	WB			

Next time

- Control hazards
  - How can the pipeline handle branch and jump instructions?