

Department of Electrical and Computer Engineering


Course Number	EE8207
Course Title	High Performance Computer System Design
Semester/Year	Winter/2016

Instructor	Dr. Nagi N. Mekhiel
------------	---------------------

Lab No.	3
----------------	----------

Assignment Title	Data Hazards in MIPS Pipelines
------------------	--------------------------------

Submission Date	11 th March 2016
Due Date	11 th March 2016

Student Name	Muhammad Obaidullah
Student ID.	500671408
Signature*	

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.

1 OBJECTIVES OF THE LAB

1. Analyzing MIPS Pipeline using WINMIPS Simulator.
2. Demonstrate Data Hazards.
3. Performance cost of Data Hazards.
4. Reduce penalty of Data Hazards

2 DOWNLOADING & SETTING UP WINMIPS64

2.1 INSTALLATION

To install WinMIPS64, following steps were taken:

1. WinMIPS64 was downloaded from <http://indigo.ie/~mscott/> [1]
2. After downloading, winmips64.exe file was opened.
3. A folder name "Codes" was created for saving all codes and running in the same directory.

2.2 ANALYSES

1. A test assembler program was written as follows:

```
1      .data
A:      .word 10
3      B:      .word 8
C:      .word 0
5
      .text
7      main:
      ld  r4 ,A(r0)
9      ld  r5 ,B(r0)
      dadd r3 ,r4 ,r5
11     sd  r3 ,C(r0)
      halt
```

2. *.data* signifies the start of data segment where we can set some data in the memory before the execution of the program starts.
3. *.text* signifies the start of text segment where we write our program.
4. *.main* signifies the start of main program.
5. The test program takes two numbers and adds them.
6. By selecting File > Open, we can run the assembly code and by pressing F7 we can step in code and see the pipeline.

3 DATA HAZARDS

3.1 READ AFTER WRITE (RAW)

This type of hazard occurs when an instruction tries to read a register before a previous instructions writes an update value to it. [2]

3.1.1 WITHOUT FORWARDING

Hazard: RAW without forwarding **Cost:** 2 cycles

The following code was written:

```

0000 dc040000 ld r4,A(r0)
0004 dc050008 ld r5,B(r0)
0008 0085182c dadd r3,r4,r5
000c fc030010 sd r3,C(r0)
0010 04000000 halt
0014 00000000
    
```

Figure 3.1: Assembly code for adding two numbers A and B and then storing C.

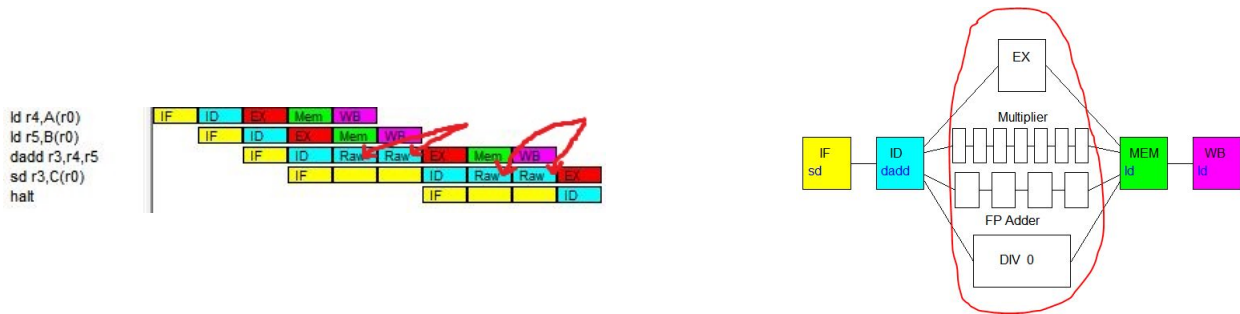


Figure 3.2: Cycles view showing two cycles of stall when there is RAW hazard while pipeline view showing execution block doing no work

3.1.2 WITH FORWARDING

Hazard: RAW with forwarding **Cost:** 1 cycles

In WinMIPS simulator Configure > Enable Forwarding was turned ON. This allows the values after execution stage to be forwarded to the next execution so that it can immediately start next instruction execution. This way penalty is reduced from 2 cycles stall to 1 cycle stall.

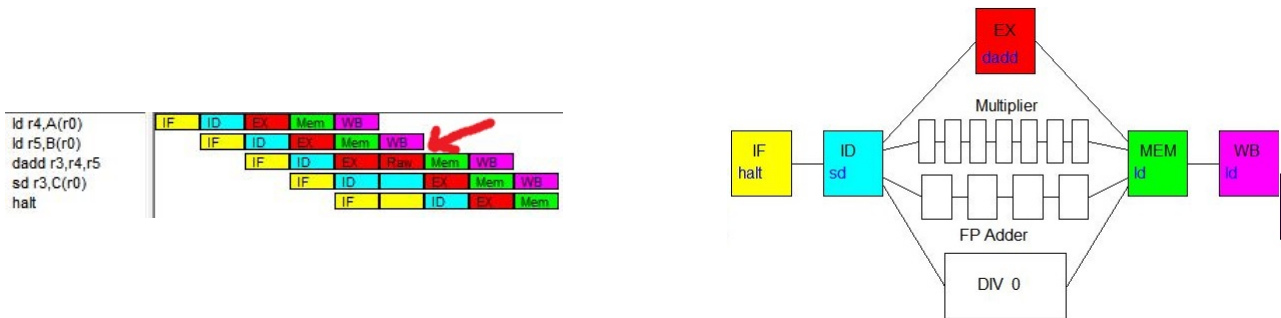


Figure 3.3: Cycles view showing only 1 cycle of stall when there is RAW hazard while pipeline view showing execution block ready to execute add instruction.

3.1.3 OPTIMAL REDUCED PENALTY SOLUTION

Hazard: RAW with scheduling and forwarding **Cost:** 0 cycles

The original code was modified to include one instruction between load and add as follows:

```

0000 dc040000 ld r4,A(r0)
0004 dc050008 ld r5,B(r0)
0008 dc060018 ld r6,D(r0) ← New
000c 0085182c dadd r3,r4,r5
0010 fc030010 sd r3,C(r0)
0014 04000000 halt
0018 00000000

```

Figure 3.4: Assembly code for adding two numbers A and B and then storing C.

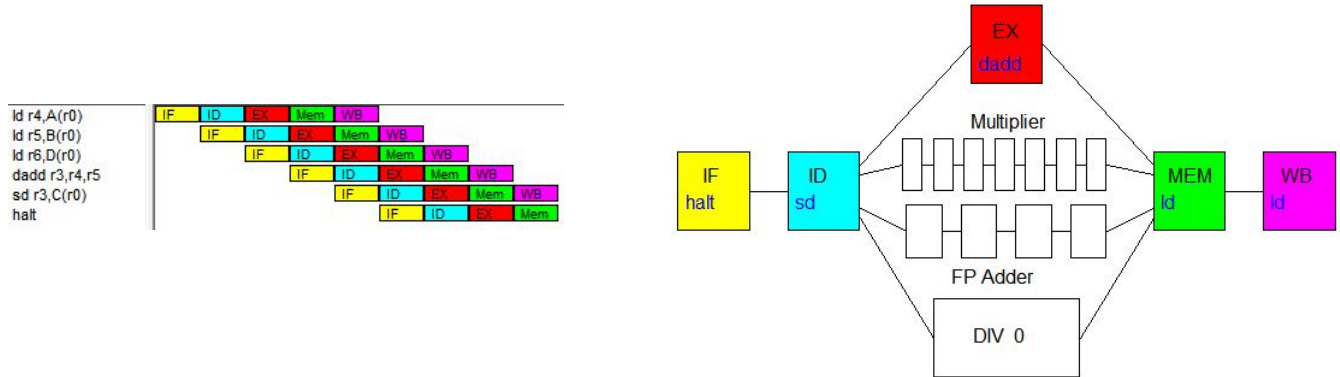


Figure 3.5: Cycles view showing no stall while pipeline view showing every block busy.

3.2 WRITE AFTER READ (WAR)

This type of hazard occurs when an instruction tries to write to a register before a previous instructions has read its value.

3.2.1 PROBLEM

Hazard: WAR **Cost:** 3 cycles

The following code was written:

```

0000 d4070000 l.d f7,A(r0)
0004 d4030008 l.d f3,B(r0)
0008 d4040018 l.d f4,D(r0)
000c 462339c0 add.d f7,f7,f3
0010 462439c0 add.d f7,f7,f4
0014 46262902 mul.d f4,f5,f6 ←
0018 f4070010 s.d f7,C(r0)
001c f4040018 s.d f4,D(r0)
0020 04000000 halt

```

Figure 3.6: Assembly code for adding three floating points A, B, and C.

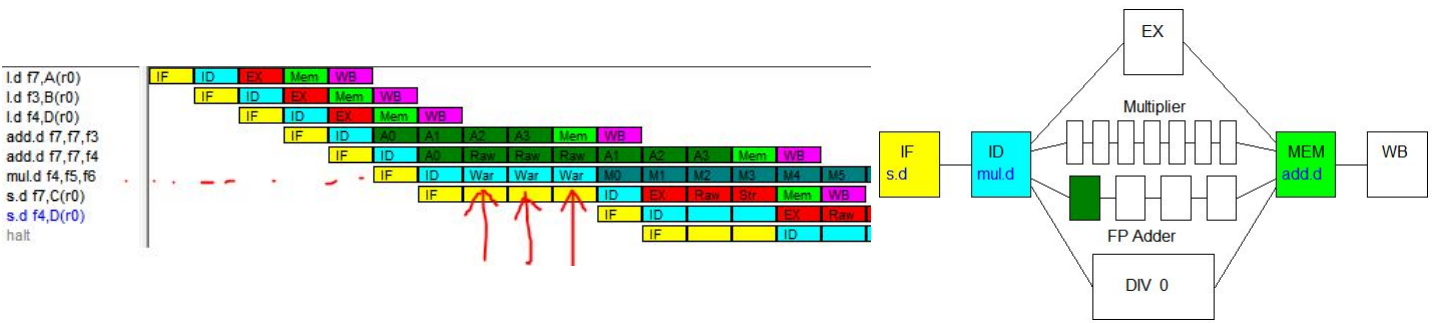


Figure 3.7: Cycles view showing 3 cycles of stall when there is WAR hazard. If the mul.d is allowed to issue, it could "overtake" the second add.d and write to f4 first. Therefore in this case the mul.d must be stalled in ID.

3.2.2 SOLUTION

WAR data hazard can be removed by register renaming.

Hazard: WAR with register renaming **Cost:** 0 cycles

The code was modified to following:

```

0000 d4070000    l.d f7,A(r0)
0004 d4030008    l.d f3,B(r0)
0008 d4040018    l.d f4,D(r0)
000c 462339c0    add.d f7,f7,f3
0010 462439c0    add.d f7,f7,f4
0014 46262a02    mul.d f8,f5,f6
0018 f4070010    s.d f7,C(r0)
001c f4080018    s.d f8,D(r0)
0020 04000000    halt
  
```

Figure 3.8: Assembly code for adding three floating points A, B, and C.

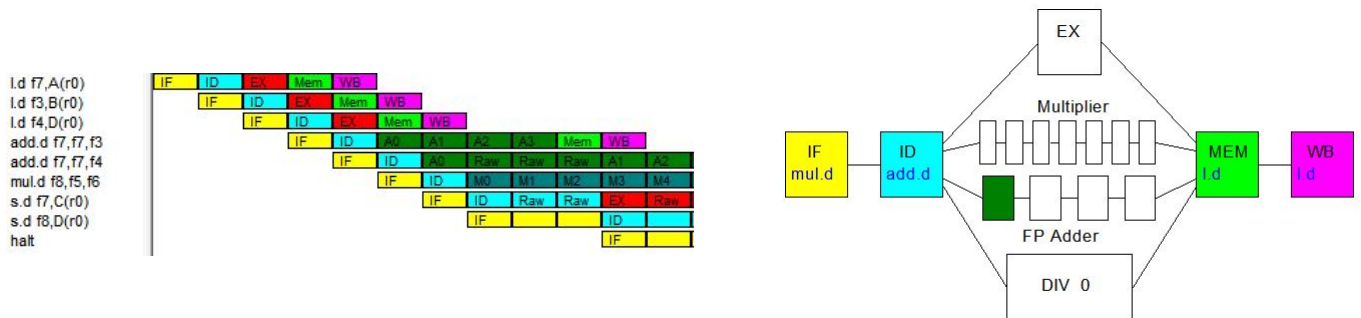


Figure 3.9: Cycles view showing 3 cycles of stall when there is WAR hazard. If the mul.d is allowed to issue, it could "overtake" the second add.d and write to f4 first. Therefore in this case the mul.d must be stalled in ID.

3.3 WRITE AFTER WRITE (WAR)

This type of hazard occurs when an instruction tries to write to a register immediately after previous instructions has written its value. This type of hazard can occur more often in out-of-order execution of instructions.

3.3.1 PROBLEM

Hazard: WAW **Cost:** 1 cycle

The following code was written:

```

0000 d4070000    l.d f7,A(r0)
0004 d4030008    l.d f3,B(r0)
0008 d4040018    l.d f4,D(r0)
000c 462339c2    mul.d f7,f7,f3
0010 46262900    add.d f4,f5,f6
0014 46262900    add.d f4,f5,f6
0018 462321c0    add.d f7,f4,f3
001c f4070010    s.d f7,C(r0)
0020 f4040018    s.d f4,D(r0)
0024 04000000    halt

```

Figure 3.10: Assembly code for adding three floating points A, B, and C.

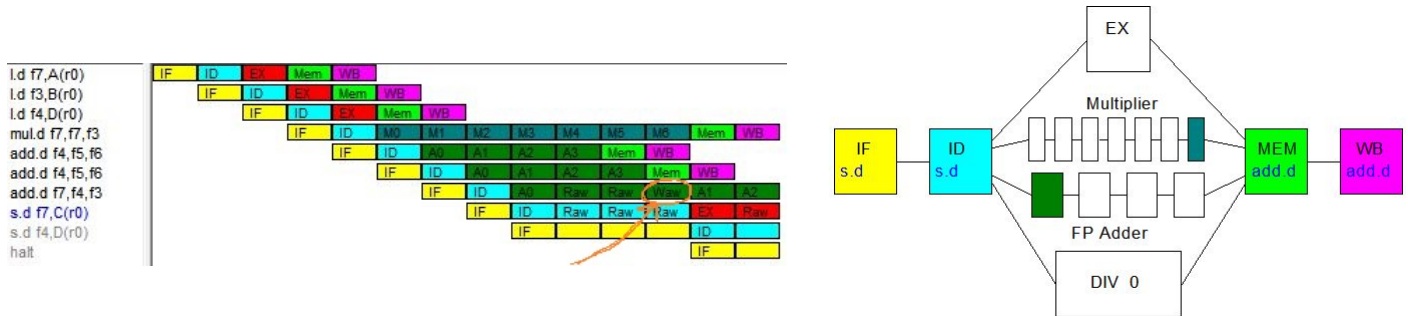


Figure 3.11: Cycles view showing 1 cycle of stall when there is WAW hazard.

3.3.2 SOLUTION

Hazard: WAW with register renaming **Cost:** 0 cycle

The code was modified to following where the register f7 was renamed to f8:

```

0000 d4070000    l.d f7,A(r0)
0004 d4030008    l.d f3,B(r0)
0008 d4040018    l.d f4,D(r0)
000c 462339c2    mul.d f7,f7,f3
0010 46262900    add.d f4,f5,f6
0014 46262900    add.d f4,f5,f6
0018 46232200    add.d f8,f4,f3
001c f4080010    s.d f8,C(r0)
0020 f4040018    s.d f4,D(r0)
0024 04000000    halt

```

Figure 3.12: Assembly code for adding three floating points A, B, and C.

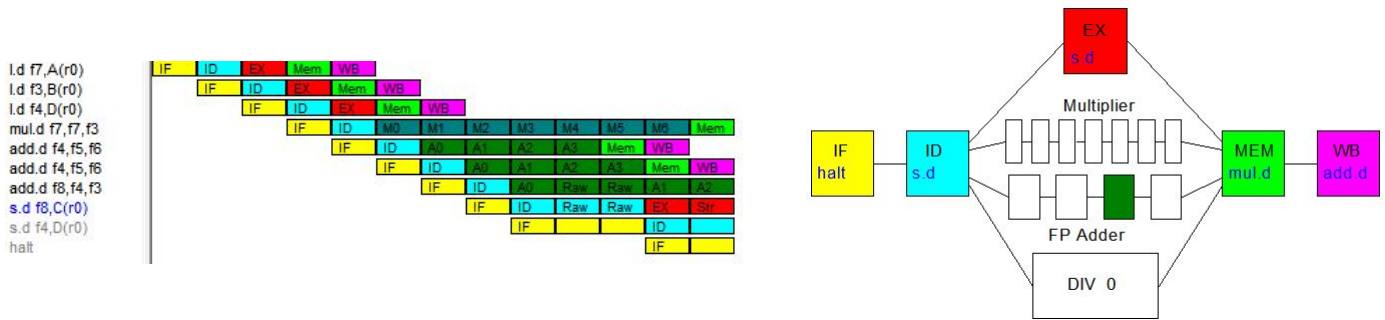


Figure 3.13: Cycles view showing 1 cycle of stall when there is WAW hazard.

4 CONCLUSION

- WAW and WAR hazards can easily be reduced to 0 cycle latency cost by combination of scheduling and register renaming.
- RAW hazard penalty can be reduced to fewer cycles by forwarding. But the penalty can completely be hidden by rescheduling the code to hide latency.
- Intelligent Compiler design is very important in order to reduce penalties and re-schedule the code after mapping data dependencies.

REFERENCES

- [1] M. Scott. (2012, April) Winmips64. [Online]. Available: <http://indigo.ie/~mscott/>
- [2] D. Tullsen. Pipeline hazards. Pdf. Jacobs School of Engineering. [Online]. Available: <http://cseweb.ucsd.edu/classes/wi05/cse240a/pipe2.pdf>