School of Engineering and Computer Science

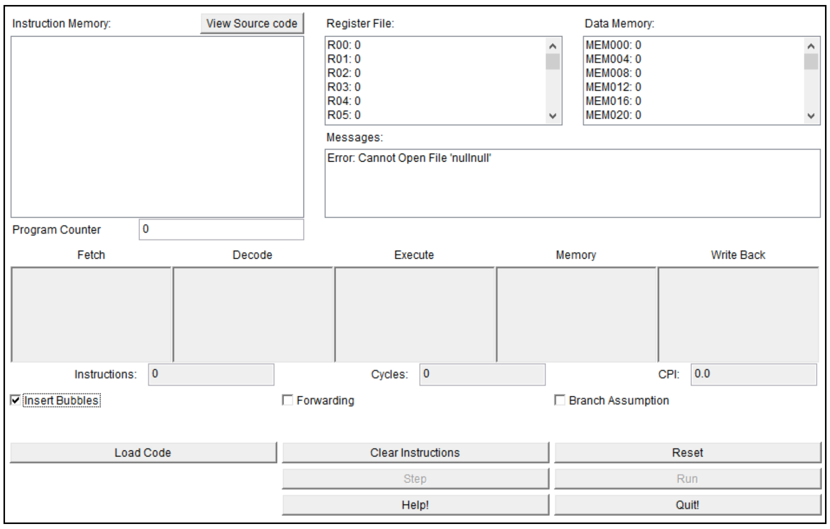
NWEN 242 **Computer Organization**

Project 2

The objective of this lab is to test your understanding of pipeline datapath operation and hazards. It is worth **10%** of your final grade. The project is **marked out of 100**.

Contents:

1. Introduction to MPSIM



R00: 0
R01: 0
R02: 0
R03: 0
R04: 0
R05: 0

MEM000: 0
MEM004: 0
MEM008: 0
MEM012: 0
MEM016: 0
MEM020: 0

Error: Cannot Open File 'nullnull'

Program Counter: 0

Instructions: 0 Cycles: 0 CPI: 0.0

[✓ Insert Bubbles] ☐ Forwarding ☐ Branch Assumption

Figure 1: Screenshot of MPSIM.

In this lab you will use a new simulator called **MPSIM**. A screenshot of MPSIM is given in Figure 1. MPSIM is written completely in Java. You can download MPSIM.jar from the project website. After downloading the jar file, you need to change your working directory to the folder that contains the jar file. Then on the command line, type the following to start the simulator.

```
% java -jar MPSIM.jar
```

You can also run MPSIM.jar at your home computer (Windows, Mac, and Linux). Make sure that your computer has the Java runtime installed.

1.1 How to use the MPSIM Pipeline simulator

Store a MIPS assembly program prepared for running by MPSIM Pipeline Simulator in your home directory. To execute a program, perform the following procedure:

- Load your program using the Load button (browse to your program if needed).
- Choose a data_path operation mode by asserting appropriate mode buttons.
- Execute your program either stepwise (using the Step button repetitively), or continuously (Clicking on the Run button).

It is advisable to step through your program first, to see how it behaves, and then to run it.

1.2 Instructions supported by the MPSIM Pipeline simulator

The MPSIM Pipeline simulator supports many MIPS instructions. They are:

- R-Type:
  - ADD, SUB, AND, OR, SLL, SRL, SLT;
- I-Type:
  - ADDI, SUBI, ANDI, ORI, SLLI, SRLI;
- Branch:
  - BEQ, BNE;
- Memory:
  - SW, LW.

It should be noted that the simulator is not case sensitive. So, "*add*" and "*ADD*" are both plausible op codes. The simulator does not allow using register zero ($0) as the destination register of arithmetic and logic instructions. This restriction is not present in the MIPS assembly language.

There are two special pseudo instructions supported. These are "*NOP*" and "*END*". The NOP pseudo instruction does nothing and should be used in programs being executed on a data path that does not insert bubbles automatically. The END pseudo instruction is used to tell the simulator that the program end has been reached.

There is also a system generated pseudo instruction "*INVALID*". The INVALID instruction is entered into the IF stage when the PC addresses a blank word in the instruction memory (i.e. a non-existent code line in the program). It is not intended for use in MIPS programs. Its only aim is to clearly designate a wrong program control.

1.3 Modes of Operation

MPSIM provides three settings: "*insert bubbles*", "*forwarding*", and "*branch assumption*". By using a combination of the three settings, MPSIM can simulate five different MIPS processor pipeline modes:

- Basic Mode (turn off all settings),
- Bubbles Mode (turn on only "insert bubbles"),

- Bubbles with Branch Assumption (turn on “insert bubbles” and “branch assumption”),
- Forwarding (turn on “forwarding”), and
- Forwarding with Branch Assumption (turn on “forwarding” and “branch assumption”).

1.4 Pipeline settings and assumptions for each mode of operations

Pipeline settings and assumptions for each mode of operations in the MPSIM Pipeline simulator is not completely identical to the pipeline technologies discussed in the lecture and in the textbook. Therefore, it is VERY IMPORTANT for you to read and understand this subsection.

1.4.1 Register file settings

During *Basic mode* (i.e. when both Bubbles and Forwarding buttons are off), MPSIM simulates a pipeline datapath where the Register File (RF) is *read in the first half* and *written in the second half* of a processor cycle.

Otherwise, when either Bubbles or Forwarding is on, MPSIM simulates a pipeline data_path where the Register File (RF) is *written in the first half* and *read in the second half* of a processor cycle.

1.4.2 Basic mode

When all three mode buttons are off, the program simulates the basic pipeline data_path that relies completely on the compiler (or student) control of hazards and has no performance enhancements in the case of hazards.

The number of *nops* needed between an R-type instruction (i_1) and a consecutive instruction (i_2) needing its result data is computed by subtracting the stage number when the instruction i_2 needs data from the stage number when the instruction i_1 writes data into the register File.

The number *nops* needed after a branch instruction is computed by subtracting 1 from the stage number when the branch instruction updates the PC.

1.4.3 Bubbles mode and Branch Assumption is off

When the Bubbles button is on, the program simulates a pipeline datapath that does hazard detection, inserts bubbles, and stalls the pipeline.

The *branch decision is made* in the *third (EX) stage*. The *branch target address is calculated in the fourth (MEM) stage*, and if the branch taken, the *PC is also updated* in the *fourth (MEM) stage*.

The branch instruction op code in the second (ID) stage induces a pipeline stall, the simulator stops the PC from advancing. To avoid reading instructions that perhaps should not be executed, and *inserts three bubbles* in the form of *nops* after the branch instruction.

1.4.4 Bubbles mode and Branch Assumption is on

When the Bubbles button is on, the program simulates a pipeline datapath that does hazard detection, inserts bubbles, and stalls the pipeline.

The *branch decision is made in the third (EX) stage*. The *branch target address is calculated in the fourth (MEM) stage*, and if the branch taken, the *PC is also updated in the fourth (MEM) stage*.

"*Branch assumption on*" means that a branch not taken assumption is made. So, no bubbles are inserted after a branch instruction. If a branch is not taken, the instruction following it will be executed. If the branch is taken, in cycle *c*, then:

- PC will be added by 4 at the start of the cycle *c+1,*
- IF, ID and EX stages will also be ***flushed*** at the start of the cycle *c+1,*
- PC will be updated to the target address at the end of the cycle *c+1, and*
- The instruction at the target address will be fetched in the cycle *c+2.*

1.4.5 Forwarding mode and Branch Assumption is off

Note that, if the Forwarding button is on, *Bubbles button has to be off.* When the Forwarding button is on, MPSIM simulates a pipeline datapath that makes *branching decision and possible Program Counter (PC) update within the second (ID) stage*, and *does not insert any bubbles and does not stall the pipeline*. The *branch target address is calculated in the second (ID) stage*.

In the case that an instruction just preceding a branch computes the data needed for comparison and decision making, which the data will be available at the end of EX stage, the hazard detection is capable of detecting the need for data being in the EX/MEM pipeline register at the input of comparator's XOR gates. Also, *the forwarding unit is capable of supplying data from EX/MEM pipeline register to the comparator*.

1.4.6 Forwarding mode and Branch Assumption is on

Note that, if the Forwarding button is on, *Bubbles button has to be off.* When the Forwarding button is on, MPSIM simulates a pipeline datapath that makes *branching decision and possible Program Counter (PC) update within the second (ID) stage*, and *does not insert any bubbles and does not stall the pipeline*. The *branch target address is calculated in the second (ID) stage*.

In the case that an instruction just preceding a branch computes the data needed for comparison and decision making, which the data will be available at the end of EX stage, the hazard detection is capable of detecting the need for data being in the EX/MEM pipeline register at the input of comparator's XOR gates. Also, *the forwarding unit is capable of supplying data from EX/MEM pipeline register to the comparator*.

When the Branch Assumption button is on, MPSIM *simulates a branching strategy where the branch not taken assumption is made*. If a branch is actually taken, an appropriate number of *instructions following the branch instruction in the pipeline will be flushed*. The instructions flushed will not be executed and thus not counted.

1.4.7 Summary

A summary of different modes of operations supported by the simulated pipeline is presented in Table 1.

	Processor Cycle	Inserting NOPs	Branch Decision Making	branch target address calculation
Basic Mode	Register File (RF) is *read in the first half* and *written in the second half*	Manually	Third (EX) stage	Fourth (MEM) Stage
Bubbles +Branch Assumption off	Register File (RF) is *written in the first half* and *read in the second half*	Automatically	Third (EX) stage	Fourth (MEM) Stage
Bubbles +Branch Assumption on	Register File (RF) is *written in the first half* and *read in the second half*	Automatically	Third (EX) stage	Fourth (MEM) Stage
Forwarding +Branch Assumption off	Register File (RF) is *written in the first half* and *read in the second half*	Manually	Second (ID) Stage	Second (ID) Stage
Forwarding +Branch Assumption on	Register File (RF) is *written in the first half* and *read in the second half*	Manually	Second (ID) Stage	Second (ID) Stage

Table 1: Summary of different modes of operations in the simulated pipeline.

2. How to prepare for the lab

- Attend the lectures on pipelining (starting from week 7).
- Read the relevant chapter in the textbook (Chapter 4).
- Answer the preparatory questions.

3. Preparatory questions (not marked)

- Which types of hazards did we consider in lectures? List them and briefly describe. Give an example for each type of a hazard. Suppose there is no hardware detection and prevention of hazards.

- List and briefly describe hazard detection and prevention techniques.

- Suppose there is neither hardware hazard detection nor prevention. How many nops are needed between any two subsequent instructions being involved in a hazard?

- Suppose there is hardware detection of hazards. What feature has the special Register File, and what is the net effect of this feature?

- Describe the way the Forwarding Unit detects a hazard between an R-type instruction and an lw instruction.

- Suppose a pipelined datapath contains a forwarding unit. How many nops are needed between two consecutive R-type instructions that are at risk of a hazard?

- Suppose a pipelined datapath contains a forwarding unit. How many nops are needed between an R-type instruction and a consecutive memory instruction that is at risk of a hazard?

- Suppose a pipelined datapath contains a forwarding unit. How many nops are needed between an lw instruction and a consecutive R-type or a memory instruction that is at risk of a hazard?

- Suppose a pipelined datapath makes a decision whether to branch or not in the second stage, computes the target address in the second stage and, if the branch is taken, updates the program counter at the end of the second stage. How many nops should be inserted between an R-type instruction and a following branch instruction if the branch instruction needs the computation result of the R-type instruction?

- Suppose a pipelined datapath makes a decision whether to branch or not in the second stage, computes the target address in the second stage and, if the branch is taken, updates the program counter at the end of the second stage. How many nops should be inserted between an R-type instruction being in the MEM stage and a following branch instruction being in the ID stage, if the branch instruction needs the result of the computation performed by the R-type instruction?

- Suppose a pipelined datapath makes a decision whether to branch or not in the second stage, computes the target address in the second stage and, if the branch taken, updates the program counter at the end of the second stage. How many nops should be inserted between an lw instruction and a following branch instruction if the branch instruction needs result of reading the memory done by the lw instruction?

4. Preliminary experiments

4.1 Preliminary experiment 1

Run the following program on MPSIM with basic mode, bubbles mode and forwarding mode. **Modify the program to ensure correct operation of your program**. Check your program by single stepping through the instructions. [**10 marks**]

```
start:
addi $5, $0, 5
addi $1, $0, 1
addi $3, $0, 3
nop
nop
add $2, $5, $1
add $4, $2, $3
nop
nop
nop
END
```

Answer the following questions.

a) For each mode, after how many clock cycles will the destination register of the first add instruction, $2, receive the correct result value (in the original program, without any of your modifications)?
b) For each mode, after how many clock cycles is the value of $2 needed in the second instruction (in the original program)?
c) What is the problem here? What is this kind of hazard called?

Please name your programs according to the following:

- Basic mode: **lab2-4-1-basic.asm**
- Bubbles mode: **lab2-4-1-bubble.asm**
- Forwarding mode: **lab2-4-1-forward.asm**

Remarks on marking:

- Correctness of the program for the basic mode: **2 Marks**
- Correctness of the program for the bubbles mode: **2 Marks**
- Correctness of the program for the forwarding mode: **2 Marks**
- Correct answer to (a): **1 Mark**
- Correct answer to (b): **1 Mark**
- Correct answer to (c): **2 Marks**

4.2 Preliminary experiment 2

Run the following program on MPSIM with basic mode, forwarding, and forwarding with branch assumption. **Modify the program to ensure correct operation of your program**. Check your program by single stepping through the instructions. [**10 marks**]

```
start:
nop
nop
beq $2, $1, start
addi $2, $2, 1
nop
nop
nop
END
```

Answer the following questions.

a) For each mode, how many cycles does it take until the branch instruction is ready to jump (in the original program, without any of your modifications)?

b) What has happened with the following addi instruction while the branch is calculated (in the original program)?
c) What is the problem here? What is this kind of hazard called?

Please name your programs according to the following:

- Basic mode: **lab2-4-2-basic.asm**
- Forwarding mode: **lab2-4-2-forward.asm**
- Forwarding with branch assumption: **lab2-4-2-forward-branch.asm**

Remarks on marking:

- Correctness of the program for the basic mode: **2 Marks**
- Correctness of the program for the forwarding mode: **2 Marks**
- Correctness of the program for the forwarding with branch assumption mode: **2 Marks**
- Correct answer to (a): **1 Mark**
- Correct answer to (b): **1 Mark**
- Correct answer to (c): **2 Mark**

4.3 Preliminary experiment 3

Run the following program on MPSIM with basic mode, bubbles mode, and forwarding mode. **Modify the program when necessary to ensure correct operation of your program**. Check your program by single stepping through the instructions. [**10 marks**]

```
start:
addi $2, $0, 3
sw   $2, 0($0)
nop
nop
nop
lw   $3, 0($0)
addi  $4, $3, 2
nop
nop
nop
END
```

Answer the following questions.

a) For each mode, after how many clock cycles will the destination register of the load instruction, \$3, receive the correct result value (in the original program, without any of your modifications)?
b) For each mode, after how many clock cycles is the value of \$3 needed in the addi instruction (in the original program)?
c) What is the problem here? What is this kind of hazard called?

Please name your programs according to the following:

- Basic mode: **lab2-4-3-basic.asm**
- Bubbles mode: **lab2-4-3-bubble.asm**
- Forwarding mode: **lab2-4-3-forward.asm**

Remarks on marking:

- Correctness of the program for the basic mode: **2 Marks**
- Correctness of the program for the bubbles mode: **2 Marks**
- Correctness of the program for the forwarding mode: **2 Marks**
- Correct answer to (a): **1 Mark**
- Correct answer to (b): **1 Mark**
- Correct answer to (c): **2 Mark**

5. Modify and optimize a program

Copy the following assembly code into a file and name it **lab2program.asm**.

```
addi $2, $0, 3
sw   $2, 0($0)
addi $3, $0, 2
addi $9, $0, 12
sw   $9, 12($0)
uncon:
add  $1, $2, $3
lw   $5, 12($0)
sw   $1, 0($0)
addi $4, $0, 100
slt  $6, $1, $4
beq  $6, $0, fin
add  $2, $2, $2
add  $3, $3, $3
lw   $8, 0($0)
beq  $8, $0, fin
lw   $7, 0($5)
add  $7, $3, $7
beq  $0, $0, uncon
fin:
END
```

5.1 Make appropriate changes to the program lab2program.asm to run it in: [**30 marks**]

d) Basic Mode (all mode option buttons off),
e) Bubbles Mode (**Bubbles** button on),
f) Bubbles with Branch Assumption (**Bubbles**, **Branch Assumption** buttons on),
g) Forwarding (**Forwarding** button on), and
h) Forwarding with Branch Assumption (**Forwarding**, **Branch Assumption** buttons on).

To make changes, **do not optimize your program** by reordering or even omitting particular instructions, only insert an appropriate number of **NOPs** where needed.

In the program, the register **$7** accumulates result. Use the content of the register $7 to test whether your program runs correctly or not. When the immediate of the first program instruction is 3, register **$7 = 76**. When it is set to 0, **$7 = 140**, use this to test your program.

Note: you will need to make five different tests of your program. For all tests, you will need to make a different version of the lab2program.asm. You need to submit all versions of your program. Please name your programs according to the following:

- Basic mode: **lab2program-basic.asm**
- Bubbles mode: **lab2program-bubble.asm**
- Bubbles with branch assumption: **lab2program-bubble-branch.asm**
- Forwarding mode: **lab2program-forward.asm**
- Forwarding with branch assumption: **lab2program-forward-branch.asm**

When your programs produce correct results, enter appropriate data into Table 2 below (assume that the immediate of the first program instruction is 3):

Mode Option	Instructions	Cycles	CPI
Basic			
Bubbles			
Bubbles with Branch Assumption			
Forwarding			
Forwarding with Branch Assumption			

Table 2: Performance report for 5.1.

Remarks on marking:

- Correctness of the program for the basic mode: **5 Marks**
- Correctness of the program for the bubbles mode: **5 Marks**
- Correctness of the program for the bubbles with branch assumption mode: **5 Marks**
- Correctness of the program for the forwarding mode: **5 Marks**
- Correctness of the program for the forwarding with branch assumption mode: **5 Marks**
- Correct answer to Table 2: **5 Marks**

5.2 When the same program is executed in different pipeline structures, the number of cycles is a measure of the performance of a pipeline structure. Analyze the content of your "**Cycles**" column in Table 2 and answer:

a) Why is the number of cycles and not number of instructions or CPI (Ref. Table 1) a measure of the performance of a pipeline structure? [**5 Marks**]

b) Different versions of the same program are executed on different pipeline datapaths. Why do they exhibit different performance figures? [**5 Marks**]

5.3 Optimize your Forwarding and Forwarding with Branch Assumption programs by reordering instructions (no instructions should be deleted except NOP). Test whether they perform correctly by setting immediate in the first program instruction to **3** and then to **0**. Set the immediate to **0** and record the appropriate data into Table 3: [**30 marks**]

Mode Option	Instructions	Cycles	CPI
Forwarding Optimized			
Forwarding with Branch Assumption Optimized			

Table 3: Performance report for 5.3.

You need to submit your optimized programs and **highlight through comments all the optimizations you have introduced in your programs**. Please name your programs according to the following:

- Forwarding mode: **lab2program-opt-forward.asm**
- Forwarding with branch assumption: **lab2program-opt-forward-branch.asm**

Remarks on marking:

- The optimized program under the forwarding mode produces correct output: **5 Marks**
- The optimized program under the forwarding with branch assumption mode produces correct output: **5 Marks**
- The optimized program under the forwarding mode introduced **at least three meaningful optimizations** (e.g. changed the position of an instruction that results in reduced instruction count, removed a nop by changing the position of an instruction): **8 Marks**
- The optimized program under the forwarding with branch assumption mode introduced **at least three meaningful optimizations** (e.g. changed the position of an instruction that results in reduced instruction count, removed a nop by changing the position of an instruction): **8 Marks**
- Performance of the optimized programs is tested and recorded in Table 3: **4 Marks**

6. How and what you should submit?

- Submit electronically before **Sunday 4 October 2015** at **23:59**. The online submission link can be found by clicking here.
- Submit your answers to questions 4.1, 4.2, 4.3, 5.1, 5.2, and 5.3 in a **PDF document**.
- Submit all your assembly programs.