

CSCI 2500 — Computer Organization
Homework 5 (document version 1.0)
Pipelining in MIPS

Overview

- This homework is due by 11:59:59 PM on Tuesday, November 19, 2019.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment, and your code **must** successfully execute on Submittly to obtain full credit.

Homework Specifications

For this individual homework assignment, you will work on a mixture of textbook problems and C code.

First, start with some “warm-up” exercises, which you will **not** submit as part of this assignment. In other words, do these “warm-up” exercises as practice and to prepare to work on the actual problems you will submit for credit.

Warm-Up Exercises (for Practice)

1. Textbook Problem 4.1 (all sub-parts)
2. Textbook Problem 4.2 (all sub-parts)
3. Textbook Problem 4.5 (all sub-parts)

Homework Problems (to Submit for Credit)

Use whatever software you like to write your answers to the textbook problems below. You must produce a PDF to submit for this assignment. Please name your PDF `hw5.pdf`. These will be manually graded by our TAs.

1. Textbook Problem 4.3 (all sub-parts)
2. Textbook Problem 4.4 (all sub-parts)
3. Textbook Problem 4.7 (all sub-parts)
4. Textbook Problem 4.8 (sub-parts 4.8.1 - 4.8.5)

Coding Problem (to Submit for Credit)

You will use C to implement a simulation of MIPS pipelining. As we've covered in lecture, there are five stages to the pipeline, i.e., IF, ID, EX, MEM, and WB.

For your simulation, you are required to support the `add`, `addi`, `sub`, `and`, `andi`, `or`, `ori`, `lw`, and `sw` instructions. More specifically, you must simulate (and output) how a given sequence of instructions would be pipelined in a five-stage MIPS implementation.

Do **not** implement forwarding in your simulation.

You can assume that each given instruction will be syntactically correct. You can also assume that there is a single space character between the instruction and its parameters. Further, each parameter is delimited by a comma or parentheses. Below are a few example instructions that you must support:

```
add $t0,$s2,$s3
addi $t1,$t3,73
or $s0,$s0,$t3
lw $a0,12($sp)
sw $t6,32($a1)
```

Required Command-Line Argument

Your program must accept one command-line argument as input. This argument (i.e., `argv[1]`) specifies the input file containing MIPS code to simulate. You may assume that no more than five instructions are given in the input file. And note that each instruction will end with a newline character (i.e., `'\n'`, ASCII code `0xa`).

Required Output

For your output, you must show *each cycle* of program execution. Each cycle will correspond to a column of output. Initially, each column is empty, indicated by a period (i.e., `'.'`). Use TAB characters (i.e., `'\t'`) to delimit each column. And assume that you will have no more than nine cycles to simulate.

Recall that a *data hazard* describes a situation in which the next instruction cannot be executed in the next cycle until a previous instruction is complete. Your code should be able to detect when it is necessary to insert one or more “bubbles” (see Section 4.7 of the textbook and corresponding lecture notes for more details).

More specifically, you will need to properly handle data hazards by adding `nop` instructions as necessary. Show these cases by indicating an asterisk (i.e., `'*'`) in the appropriate columns and adding the required number of `nop` instructions. To ensure proper formatting, add an extra TAB character after the `nop`.

On the next few pages, we present a few example runs of your program that you should use to better understand how your program should work, how you can test your code, and what output formatting to use for Submittity.

The first example (i.e., `ex01.s`) includes no data hazards.

START OF SIMULATION

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF
add \$t2,\$s0,\$s5
addi \$t4,\$s3,70

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF	ID
add \$t2,\$s0,\$s5 .	IF
addi \$t4,\$s3,70

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF	ID	EX
add \$t2,\$s0,\$s5 .	IF	ID
addi \$t4,\$s3,70 .	.	IF

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF	ID	EX	MEM
add \$t2,\$s0,\$s5 .	IF	ID	EX
addi \$t4,\$s3,70 .	.	IF	ID

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF	ID	EX	MEM	WB
add \$t2,\$s0,\$s5 .	IF	ID	EX	MEM
addi \$t4,\$s3,70 .	.	IF	ID	EX

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF	ID	EX	MEM	WB
add \$t2,\$s0,\$s5 .	IF	ID	EX	MEM	WB
addi \$t4,\$s3,70 .	.	IF	ID	EX	MEM

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$s1,\$s0,\$s0 IF	ID	EX	MEM	WB
add \$t2,\$s0,\$s5 .	IF	ID	EX	MEM	WB
addi \$t4,\$s3,70 .	.	IF	ID	EX	MEM	WB	.	.	.

END OF SIMULATION

The second example (i.e., ex02.s) includes a dependency on register \$t1.

START OF SIMULATION

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF
addi \$t2,\$s0,42
addi \$t4,\$t1,70

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID
addi \$t2,\$s0,42 .	IF
addi \$t4,\$t1,70

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID	EX
addi \$t2,\$s0,42 .	IF	ID
addi \$t4,\$t1,70 .	.	IF

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID	EX	MEM
addi \$t2,\$s0,42 .	IF	ID	EX
addi \$t4,\$t1,70 .	.	IF	ID

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID	EX	MEM	WB
addi \$t2,\$s0,42 .	IF	ID	EX	MEM
nop	.	.	IF	ID	*
addi \$t4,\$t1,70 .	.	IF	ID	ID

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID	EX	MEM	WB
addi \$t2,\$s0,42 .	IF	ID	EX	MEM	WB
nop	.	.	IF	ID	*	*	.	.	.
addi \$t4,\$t1,70 .	.	IF	ID	ID	EX

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID	EX	MEM	WB
addi \$t2,\$s0,42 .	IF	ID	EX	MEM	WB
nop	.	.	IF	ID	*	*	*	.	.
addi \$t4,\$t1,70 .	.	IF	ID	ID	EX	MEM	.	.	.

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
add \$t1,\$s0,\$s0 IF	ID	EX	MEM	WB
addi \$t2,\$s0,42 .	IF	ID	EX	MEM	WB
nop	.	.	IF	ID	*	*	*	.	.
addi \$t4,\$t1,70 .	.	IF	ID	ID	EX	MEM	WB	.	.

END OF SIMULATION

The third example (i.e., `ex03.s`) includes two dependencies on register `$t2`.

START OF SIMULATION

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF
and \$t4,\$t2,\$t5
or \$t8,\$t2,\$t6

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF	ID
and \$t4,\$t2,\$t5	.	IF
or \$t8,\$t2,\$t6

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF	ID	EX
and \$t4,\$t2,\$t5	.	IF	ID
or \$t8,\$t2,\$t6	.	.	IF

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF	ID	EX	MEM
nop	.	IF	ID	*
and \$t4,\$t2,\$t5	.	IF	ID	ID
or \$t8,\$t2,\$t6	.	.	IF	IF

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF	ID	EX	MEM	WB
nop	.	IF	ID	*	*
nop	.	IF	ID	ID	*
and \$t4,\$t2,\$t5	.	IF	ID	ID	ID
or \$t8,\$t2,\$t6	.	.	IF	IF	IF

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF	ID	EX	MEM	WB
nop	.	IF	ID	*	*	*	.	.	.
nop	.	IF	ID	ID	*	*	.	.	.
and \$t4,\$t2,\$t5	.	IF	ID	ID	ID	EX	.	.	.
or \$t8,\$t2,\$t6	.	.	IF	IF	IF	ID	.	.	.

CPU Cycles ==>	1	2	3	4	5	6	7	8	9
lw \$t2,20(\$a0)	IF	ID	EX	MEM	WB
nop	.	IF	ID	*	*	*	.	.	.
nop	.	IF	ID	ID	*	*	*	.	.
and \$t4,\$t2,\$t5	.	IF	ID	ID	ID	EX	MEM	.	.
or \$t8,\$t2,\$t6	.	.	IF	IF	IF	ID	EX	.	.

```

CPU Cycles ==> 1      2      3      4      5      6      7      8      9
lw $t2,20($a0)  IF      ID      EX      MEM     WB      .      .      .      .
nop             .      IF      ID      *      *      *      .      .      .
nop             .      IF      ID      ID      *      *      *      .      .
and $t4,$t2,$t5 .      IF      ID      ID      ID      EX      MEM     WB      .
or $t8,$t2,$t6  .      .      IF      IF      IF      ID      EX      MEM     .

CPU Cycles ==> 1      2      3      4      5      6      7      8      9
lw $t2,20($a0)  IF      ID      EX      MEM     WB      .      .      .      .
nop             .      IF      ID      *      *      *      .      .      .
nop             .      IF      ID      ID      *      *      *      .      .
and $t4,$t2,$t5 .      IF      ID      ID      ID      EX      MEM     WB      .
or $t8,$t2,$t6  .      .      IF      IF      IF      ID      EX      MEM     WB

END OF SIMULATION

```

Assumptions

Given the complexity of this assignment, you can make the following assumptions:

- Assume all input files are valid.
- Assume the length of `argv[1]` is at most 128 characters.
- Assume the maximum number of cycles to be 9. As shown in the examples, always output 9 cycles/columns, regardless of the fact that fewer may be required.

Submission Instructions

For this assignment, you will submit both your code and your PDF (i.e., `hw5.pdf`) with your answers to the textbook problems to submit.

Before you submit your code, be sure that you have clearly commented your code (this should not be an after-thought). Further, your code should have a clear and logical organization.

To submit your assignment (and also perform final testing of your code), please use Submittity. Note that the test cases for this assignment will be available on Submittity a minimum of three days before the due date and will include hidden test cases.

Also as a reminder, your code **must** successfully execute on Submittity to obtain credit for this assignment.