BBM431 ADVANCED COMPUTER ARCHITECTURE

Project: Data Hazard

Aim: In this project, you will write a part of a compiler for 5-stage pipelined MIPS processor's instruction set. Specifically, the program you will write prevent some type of data hazards for MIPS processors with and without forwarding logic by inserting NOPs between dependent instructions.

Definition:

Part 1 (without forwarding) : In a typical 5-stage MIPS processor, if there is no forwarding, we need at least two instructions between two dependent instructions to prevent data hazards. In Table 1, we have dependencies on the input instruction sequence between instructions 1 and 2 on register s0 as well as instructions 3 and 5 on register t0 in the given instruction sequence. Therefore, we have to place two NOPs between 1 and 2 and 1 NOP between 3 and 5 to remove data hazards as shown on the output instruction sequence.

Table 1: An example that shows data hazards and NOP insertions for the MIPS processor without forwarding.

Input instruction sequence		**Output instruction sequence**	
Instruction number	**Instructions**	**Instruction number**	**Instructions**
1	lw $s0, 0($t5)	1	lw $s0, 0($t5)
2	add $s1, $s0, $s2	2	NOP
3	addi $t0, $t1, 4	3	NOP
4	or $t3, $t4, $t5	4	add $s1, $s0, $s2
5	sw $t0, 0($t6)	5	addi $t0, $t1, 4
		6	NOP
		7	or $t3, $t4, $t5
		8	sw $t0, 0($t6)

Your program will receive the input instruction sequence as an input. It will output two things:

1. The instructions that will cause the data hazards and the names of the registers. In our example, the output will be:
   1-2 on $s0, 3-5 on $t0
2. The output instruction sequence with necessary NOPs added as shown in Table 1.

Part 2 (with forwarding): In a typical 5-stage MIPS processor, if there is forwarding, we need only one NOP after lw instruction if the following instruction is dependent to it. In Table 2, we have dependencies on the input instruction sequence between instructions 1 and 2 on register s0. Therefore, we have to place a NOP after lw instruction to remove data hazards as shown on the output instruction sequence.

Table 2: An example that shows data hazards and NOP insertions for the MIPS processor without forwarding.

Input instruction sequence		**Output instruction sequence**	
Instruction number	**Instructions**	**Instruction number**	**Instructions**
1	lw $s0, 0($t5)	1	lw $s0, 0($t5)
2	add $s1, $s0, $s2	2	NOP
3	addi $t0, $t1, 4	3	add $s1, $s0, $s2
4	or $t3, $t4, $t5	4	addi $t0, $t1, 4
5	sw $t0, 0($t6)	5	or $t3, $t4, $t5
		6	sw $t0, 0($t6)

Your program will receive the input instruction sequence as an input. It will output two things:

1. The instructions that will cause the data hazards and the names of the registers. In our example, the output will be:
   1-2 on $s0
2. The output instruction sequence with necessary NOPs added as shown in Table 2.

Part 3 – Bonus (Scheduling): In some cases, it is possible to reschedule the instructions in order to eliminate data hazards instead of adding NOPS. In this case, we need to identify some independent instructions to move it to the place of NOPs. For example, instruction 4 in Table 3 in the input instruction sequence is an independent instruction and can be places in any order. Therefore, placing this instruction after lw instruction eliminates data hazard between instruction 1 and 2. As a result, we do not need any NOP instruction. The resultant output instruction sequence is given in Table 3.

Table 3: An example that shows data hazards and NOP insertions for the MIPS processor without forwarding.

Input instruction sequence		**Output instruction sequence**	
Instruction number	**Instructions**	**Instruction number**	**Instructions**
1	lw $s0, 0($t5)	1	lw $s0, 0($t5)
2	add $s1, $s0, $s2	2	or $t3, $t4, $t5
3	addi $t0, $t1, 4	3	add $s1, $s0, $s2
4	or $t3, $t4, $t5	4	addi $t0, $t1, 4
5	sw $t0, 0($t6)	5	sw $t0, 0($t6)

Apply instruction scheduling to eliminate data hazards in a 5-stage pipelined MIPS processor with data forwarding (i.e., you do not need to implement it with the one that does not have forwarding.) In order to do that, you need to identify all the instructions that are dependent to each other. Sometimes, groups of instructions are depended among them while another group is depended among them while these two groups are depended to each other. In this case, you can still change the order of instructions without any violation.

Notes:

1- You can use any programming language (python, java, C/C++ etc.) to implement your algorithms.
2- Assume you have only four R-type instructions (add, sub, and, or), addi, lw, and sw. You do not need to deal with branch and jump instructions.
3- Your code will take the instruction numbers followed by the instructions as a text file. The last line will be the word “end”. An example is as follows:
   1 add $s0, $s1, $s2
   2 sub $s3, $s0, $t0
   3 lw $t0, 0($0)
   4 end

What to hand in:

1. Report: Write a report of your project. In your project, explain what you have done. For each part, give two example outputs. One with the example given above and one with your own example with at least 8 instructions. If you use latex to write your report, you will get 5 points extra. You can use www.overleaf.com to write latex documents.
2. Your code.
3. Demo: Each of you will make a demo to me in the date that will be announced after submitting your report.

Submission and due date:

1. Zip your files into a single file with your name and number as the name of your file.
2. Submit it to the email address **aca.odev@gmail.com**
3. The last day to submit your project is **December 6th, 2020.**

Grading:

1. A good report: 10%
2. Correctly running code:
    a. Part 1: 60%
    b. Part 2: 30%
3. Bonus: 20%
4. Latex: 5%