BBM431 Advanced Computer Architecture
Fall 2020
Project: Data Hazard
MIPS

İbrahim Burak Tanrıkulu, 21827852

December 10, 2020

1 Introduction

In this assignment, i fetched instructions and inserted NOPs in real-time. For doing that, i used a queue with size of 2 (Reason of queue's size is; without forwarding, registers can be usable 2 cycles after executing an instruction). When an instruction fetched, destination register of that instruction will be putted into a queue. Next time we fetched instruction, firstly we will look for dependencies with this queue. If there is dependency, then we insert NOPs. Also i used string variables named "output" and "dataHazards" to printing our outputs in order.

2 Without Forwarding

In this part, we firstly fetch first instruction. Secondly with "setDependency" method, we adding first instruction's destination register to our queue. After that, we are fetching other instructions in order and execute them one by one. In this execution stage firstly we are checking dependencies with "checkDependency" method. If there is dependency, program will insert NOPs.
Dependency setting:
We have 3 conditions while dependency setting.
1- Our instruction is "sw": sw instruction doesn't create any dependency. So we are not putting anything to queue.
2- Our queue is empty: In this case, we understand that previous 2 instructions are dependent and we inserted 2 NOPs. We know that without forwarding dependencies causes 2 NOPs. So we will add destination register twice to our queue. Also i added extra "D" letter for determining that we added same register twice. Reason of adding twice is we will pop registers at every fetch and we must ensure that these registers stay at this queue.
3- Our queue is not empty: In this case, we understand that previous 2 instructions are independent. We will pop one register from queue and check, push destionation register to queue. Conditions below explain everything but lets give an example:

```
                                    QUEUE
lw $s0, 0($t5)                      [s0D][s0D]
NOP                                 [s0D]
NOP                                 []
add $s1, $s0, $s2                   [s1D][s1D]
addi $t0, $t1, 4                    [s1][t0]
or $t3, $t4, $t5                    [t0][t3]
NOP                                 [t3]
sw $t0, 0($t6)                      []
end
```

3 With Forwarding

This part is similar with first part. But there are differences too. In this part, our queue size is 1. Because there is forwarding now. In fact, we must do queue's size 2, but i dont want to push and pop registers in every cycle. In short, we will check our instruction; if it is "lw", then we push destination register to queue and check this at next instruction.

4 Scheduling

Scheduling is harder and different than previous parts. I used new methods for this part. Let me explain algorithm.
Find an instruction which is dependent to previous instructions.
Search instructions which are dependent to this instruction and are prior to this instruction.
Thirdly execute this instructions and remove from instruction list.
If previous instruction is "lw", then find independent instruction to between previous and this insruction.
Execute these 3 instructions and remove from instruction list.
Restart this until instruction list is empty.
Search algorithm:
Firstly search for independent instruction which is prior to this instruction.
If you can't find, then create a dependency set and set dependency of this instruction.
Look for next instruction, if dependent then add dependency of this instruction to set too.
Repeat until find. Set will grow up in every step for prevent from disordering.
If you can't find, then add NOP.

5 Running Program

I used Java Programming language for project. Standard libraries. Even so i will share demo.
javac MipsCompiler.java
java MipsCompiler inputFilePath

6 Examples

Example 1:

```
1 lw $r1, 0($t5)
2 lw $r2, 0($t5)
3 add $r3, $r1, $r2
4 sw $r3, 0($t5)
5 lw $r1, 0($t5)
6 lw $r2, 0($t5)
7 sub $r3, $r1, $r2
8 sw $r3, 0($t5)
9 end
```

Output part 1:

```
1-3 on $r1
2-3 on $r2
3-4 on $r3
3-4 on $r3
5-7 on $r1
6-7 on $r2
7-8 on $r3
7-8 on $r3
```

```
lw $r1, 0($t5)
lw $r2, 0($t5)
NOP
NOP
add $r3, $r1, $r2
NOP
NOP
sw $r3, 0($t5)
lw $r1, 0($t5)
lw $r2, 0($t5)
NOP
NOP
sub $r3, $r1, $r2
NOP
NOP
sw $r3, 0($t5)
```

```
Output part 2:

2-3 on $r2
6-7 on $r2

lw $r1, 0($t5)
lw $r2, 0($t5)
NOP
add $r3, $r1, $r2
sw $r3, 0($t5)
lw $r1, 0($t5)
lw $r2, 0($t5)
NOP
sub $r3, $r1, $r2
sw $r3, 0($t5)

Output part Schedule:

lw $r1, 0($t5)
lw $r2, 0($t5)
NOP
add $r3, $r1, $r2
lw $r1, 0($t5)
lw $r2, 0($t5)
sw $r3, 0($t5)
sub $r3, $r1, $r2
sw $r3, 0($t5)
```

Example 2:

```
1 lw $s0, 0($t5)
2 add $s1, $s0, $s2
3 addi $t0, $t1, 4
4 or $t3, $t4, $t5
5 sw $t0, 0($t6)
6 end
```

Output part 1:

```
1-2 on $s0
1-2 on $s0
3-5 on $t0

lw $s0, 0($t5)
NOP
NOP
add $s1, $s0, $s2
addi $t0, $t1, 4
or $t3, $t4, $t5
NOP
sw $t0, 0($t6)
```

Output part 2:

```
1-2 on $s0

lw $s0, 0($t5)
NOP
add $s1, $s0, $s2
addi $t0, $t1, 4
or $t3, $t4, $t5
sw $t0, 0($t6)
```

Output part Schedule:

```
lw $s0, 0($t5)
addi $t0, $t1, 4
add $s1, $s0, $s2
or $t3, $t4, $t5
sw $t0, 0($t6)
```

Example 3:

Combination of example 1 and 2

Outputs of part1 and part2 is basically appended.
So i will just show bonus part:

```
lw $r1, 0($t5)
lw $r2, 0($t5)
lw $s0, 0($t5)
add $r3, $r1, $r2
lw $r1, 0($t5)
lw $r2, 0($t5)
sw $r3, 0($t5)
sub $r3, $r1, $r2
addi $t0, $t1, 4
or $t3, $t4, $t5
sw $t0, 0($t6)
add $s1, $s0, $s2
sw $t0, 0($t6)
```