Lab Report 3

Computer Architecture Lab

201651015 Dipansh Khandelwal

201651008 Aman Singh

4th February, 2019

Aim

To understand the basic principles of how pipelining works, including the problems of data and branch hazards. Finally, you should have an understanding of how the instructions are used to control different parts of a data path through a control unit.

Questions and Answers

Problem - 1

#include <iregdef.h>

.set noreorder

Avoid reordering instructions

.text # Start generating instructions

.global start # The label should be globally known

.end start # The label marks an entry point

Start: lui $9, 0xbf90 # Load upper half of port address

Lower half is filled with zeros

Repeat: lbu $8, 0x0($9)

Nop

Read from the input port # Needed after load

Sb #8, 0x0($9) # Write to the output port

B repeat # Repeat the read and write cycle

Nop # Needed after branch

Li $8, 0 # Clear the register

.end start # marks the end of the program

How many instructions are in different stages of their execution at the same time?

3 to 4 instructions are in different stages of their execution at the same time.

Problem - 2

Run the following program on the pipeline simulator. Assign distinct values to t0, t1, and t3, and single step through the instructions.

```
#include <iregdef.h>

.set noreorder

.text

.globl start

.ent start

start: add t2, t0, t1

add t4, t2, t3

nop

nop

nop

.end start
```

After how many clock cycles will the destination register of the first add instruction, t2, receive the correct result value?

4

After how many clock cycles is the value of t2 needed in the second instruction?

2

What is the problem here? What is this kind of hazard called?

Value of t2 is required at 2nd clock cycle whereas it is available at a 4th clock cycle. This is called the data hazard(Read After Write Operation).

Problem - 3

Run the following program on the pipeline simulator. Assign distinct values to t0 and t1, and let t2 contain the address to a memory location where you know the contents. Then single step through the instructions

```
#include <iregdef.h>

.set noreorder

.text

.globl start

.ent start

start: lw t0, 0(t2)

add t1, t1, t0

nop

nop

nop

.end start
```

After how many clock cycles will the destination register of the load instruction, t0, receive the correct result value?

After 4 clock cycles.

After how many clock cycles is the value of t0 needed in the add instruction?

After 2 cycles.

What is the problem here? What is this kind of hazard called?

Value of t0 is required at 2nd clock cycle whereas it is available at a 4th clock cycle. This is called the data hazard(Read After Write Operation).

What are the alternative solutions that can be used?

We can use forwarding or bypassing method, basic compiler pipeline scheduling, basic dynamic scheduling, dynamic scheduling and renaming, and hardware speculation.

Does the forwarding version of the simulator, Xl-script, handle the problem with the delayed load?

Yes, it resolves the problem as it fetches data of t0 before the 4th clock cycle.