Mark Noone

15319898

MANOONE@TCD.ie

Tutorial 4

COMPUTER ARCHITECTURE

CS3021

November 11, 2017

## Question 1

### Part 1

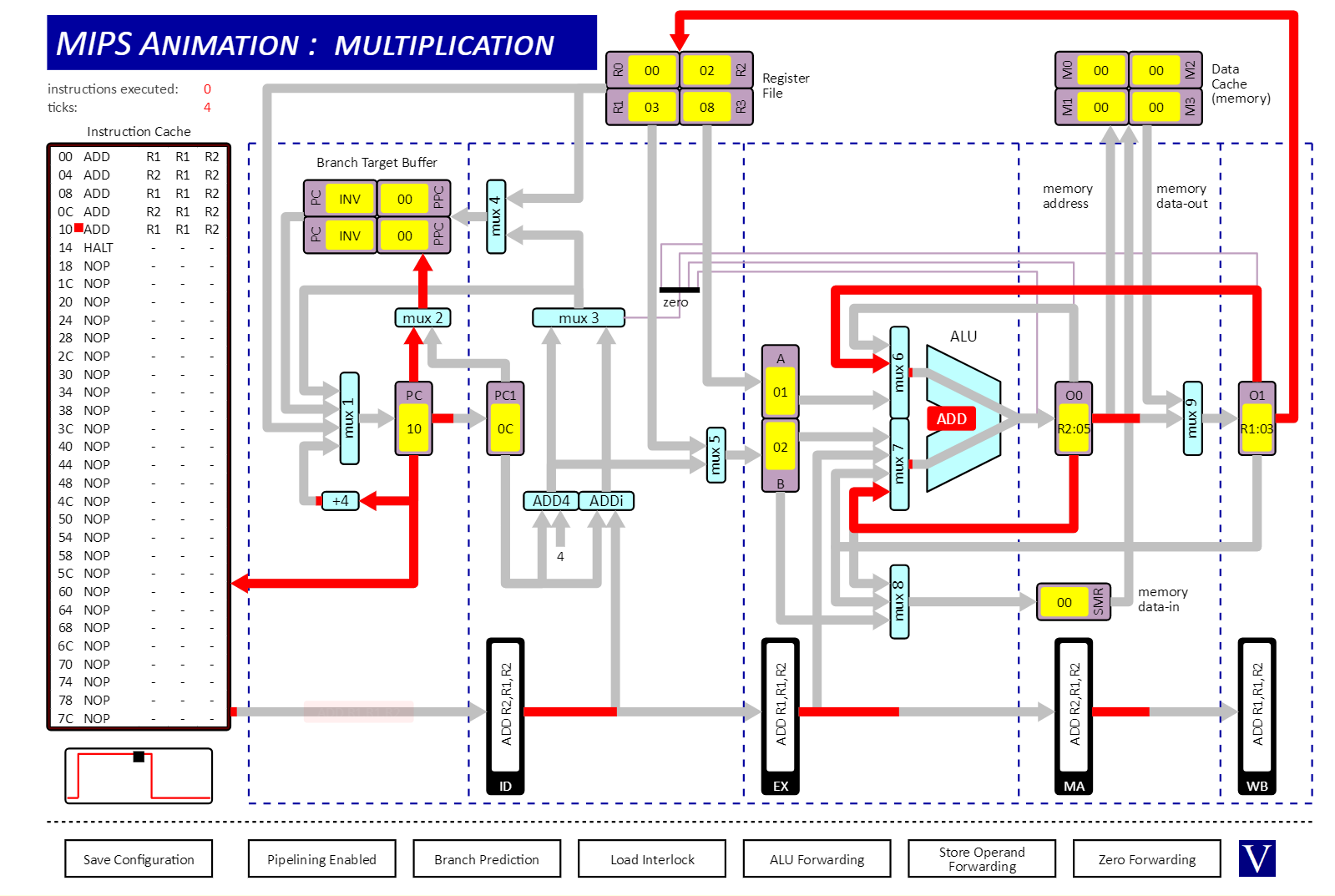
ADD R1, R1, R2

ADD R2, R1, R2

ADD R1, R1, R2

ADD R2, R1, R2

ADD R1, R1, R2



### Part 2

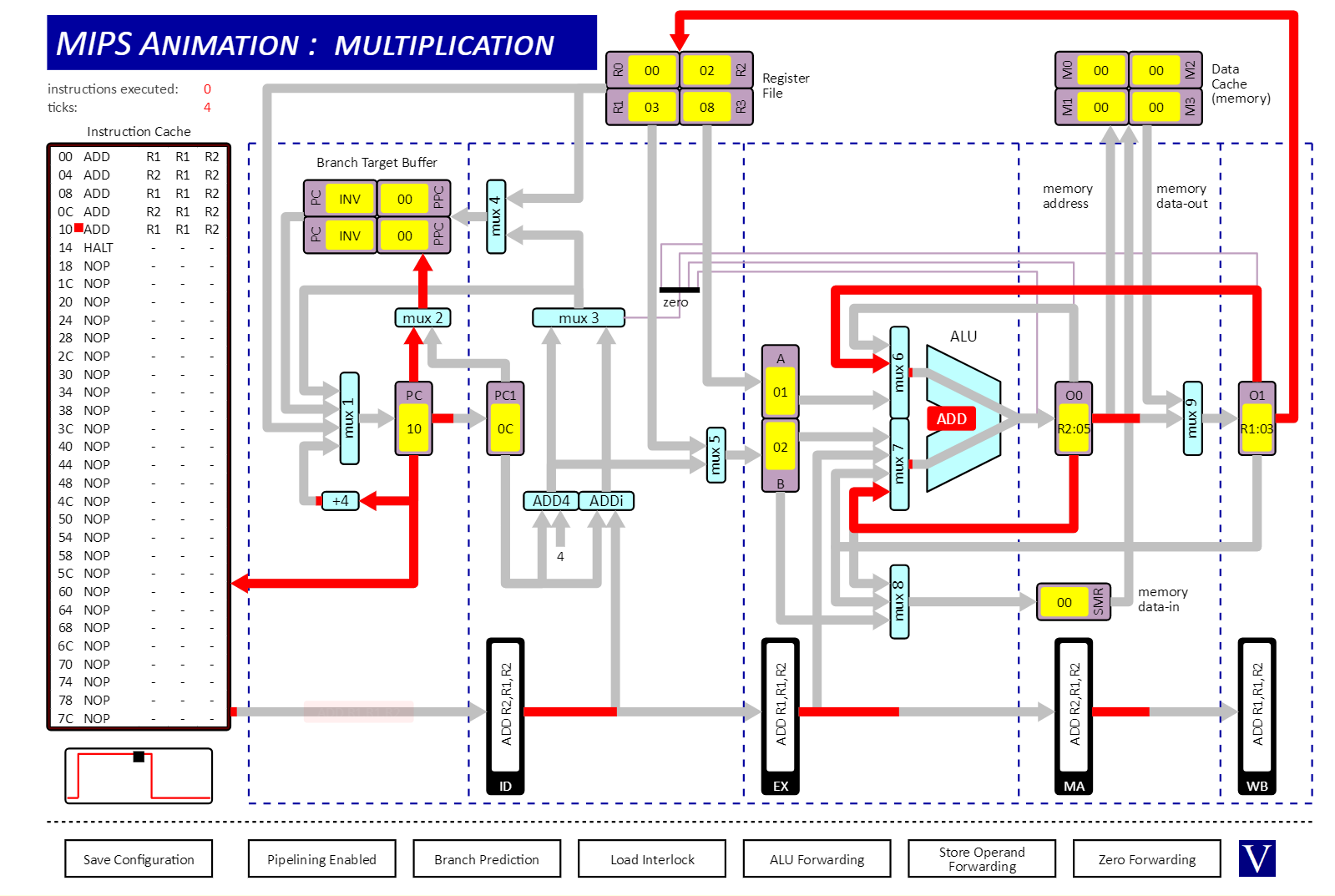
ADD R1, R1, R2

ADD R2, R1, R2

ADD R1, R1, R2

ADD R2, R1, R2

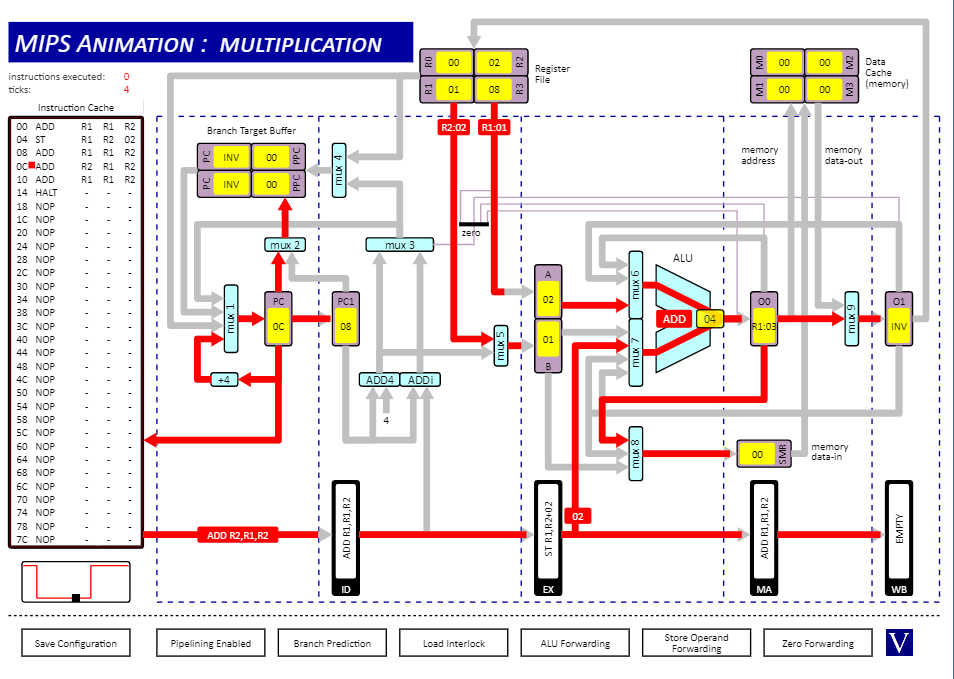
ADD R1, R1, R2



### Part 3

ADD R1, R1, R2

ST R1, R2, O2



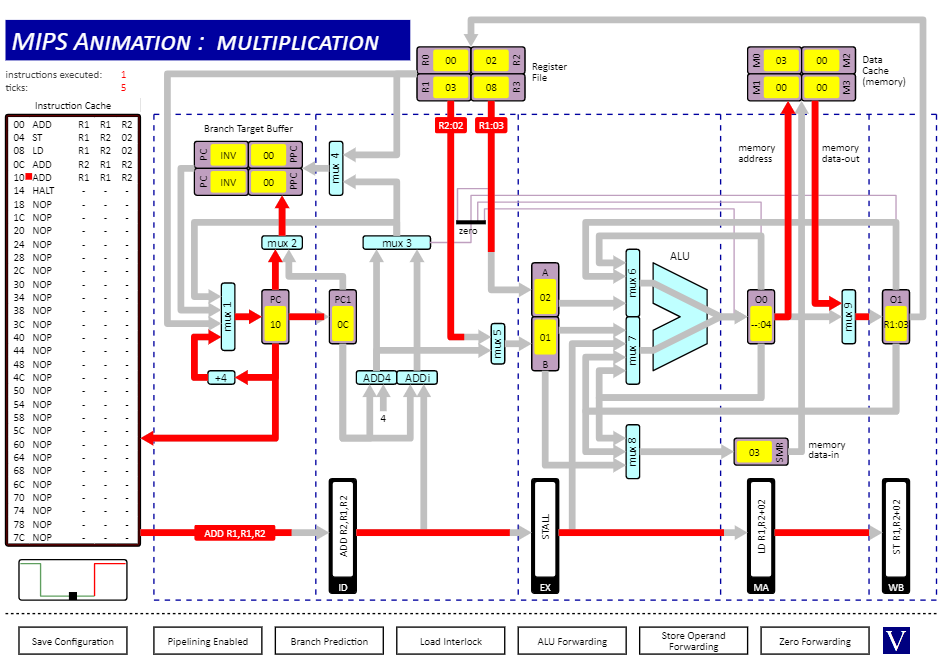
### Part 4

### Part 5

ADD R1, R1, R2

ST R1, R2, O2

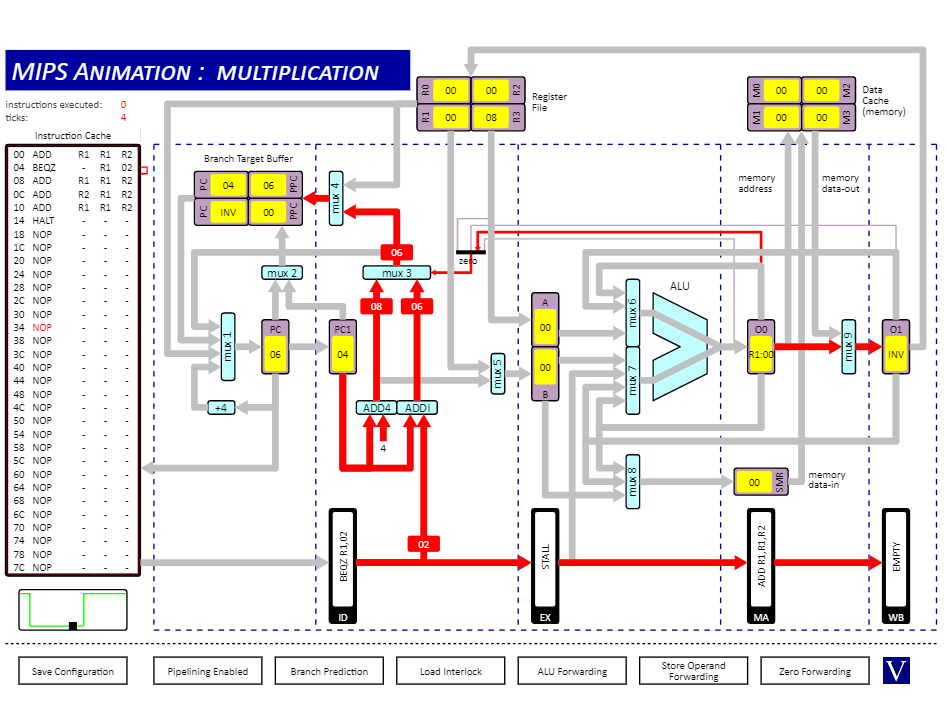
LD R1, R1, R2



### Part 6

ADD R1, R1, R2

BEQZ -, R1, O2



### Part 7

### Part 8

## Question 2

### Part 1 (alu forwarding)

Resulting Value in R1: 15

Number Of Clock Cycles: 9

### Part 2 (Alu forwarding disabled, interlocks enabled)

Resulting Value in R1: 15

Number Of Clock Cycles: 17

### Part 3 (Alu forwarding disabled, interlocks disabled)

Resulting Value in R1: 6

Number Of Clock Cycles: 9

### Part 4 (Figure explanations)

The first figures we acquire are when the ALU forwarding is turned on. This means there are no stall cycles, meaning we can achieve the best efficiency for the processor but also keep our data accurate. This is because there is a register in between the execution and memory access stages, known as ALUout 0, and then another register in between the memory access and write back stages. Because of the way our pipeline divides up the work of our instructions, we may not always have the most recent correct data stored in our registers. It takes 3 stages of the pipeline to write back our accurate results to our registers. These stages are the execution, memory access and write back stages. This means that for a whole three stages we have incorrect data in our registers which can be used by following instructions. In order to solve this issue, we use the two ALUout registers as described before to provide the most recent data sets to the ALU in the event that a register file has not yet been updated. This ability to provide the most recent information to the ALU from these temporary registers is known as ALU forwarding, and that was how it works and how we are still able to get the correct answer while keeping our cycles low.

When we turn of forwarding and leave on our interlock, the processor uses stall cycles which allow the recently changed data to make it back to the register before any other instruction that needs this data is executed. It does this by using stall cycles and the program will not progress if an instruction’s operands are not up to date and will continue to stall until data reaches registers. This is the reason for our larger number of cycles but we still have the same value.

We then have the option to turn of interlock as well and this takes away all safeguards. The program is executed no matter what and no stall cycles are used. Because of this, we get the wrong number as a result but can see that our cycles are the same as they were initially with ALU forwarding on. This is because this is the smallest amount of cycles possible with this pipeline.

## Question 3

Instructions Executed: 13

Cycles: 19