NOVEMBER 11, 2017

Tutorial 4

COMPUTER ARCHITECTURE

CS3021

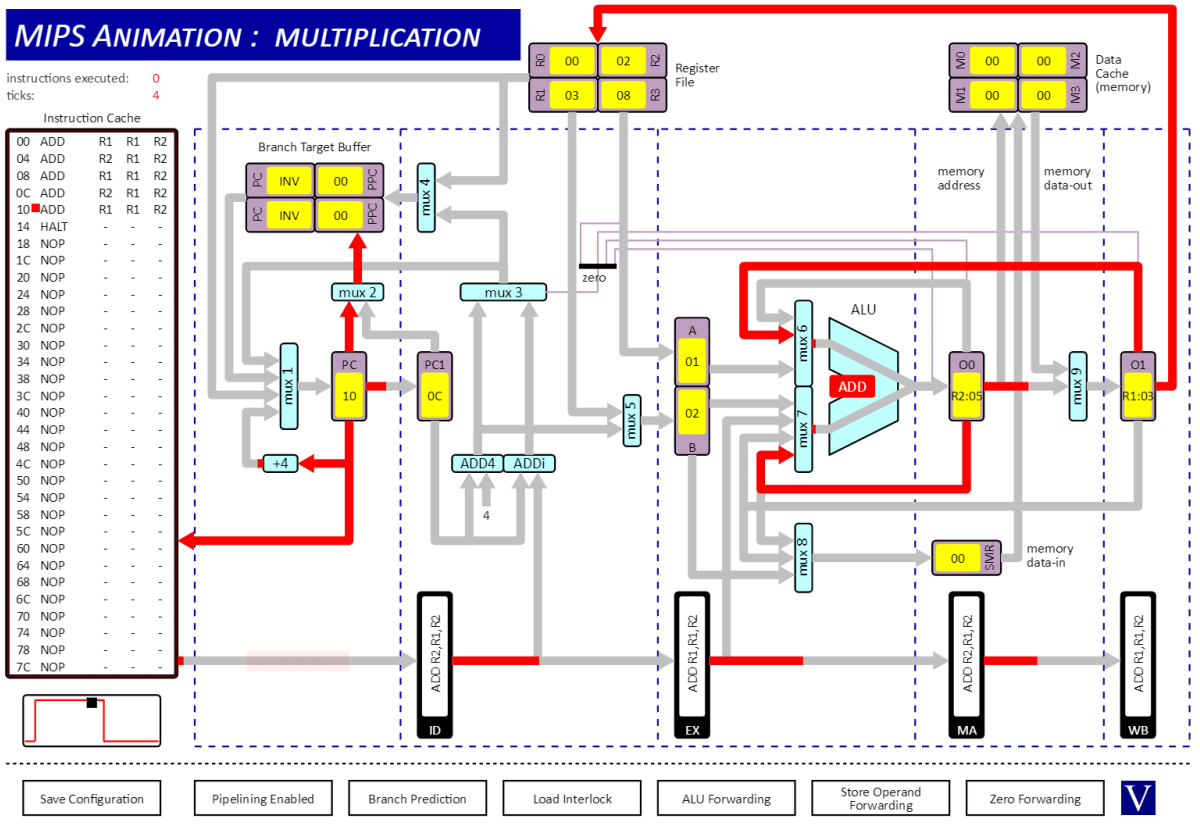
MARK NOONE

15319898
MANOONE@TCD.IE

QUESTION 1

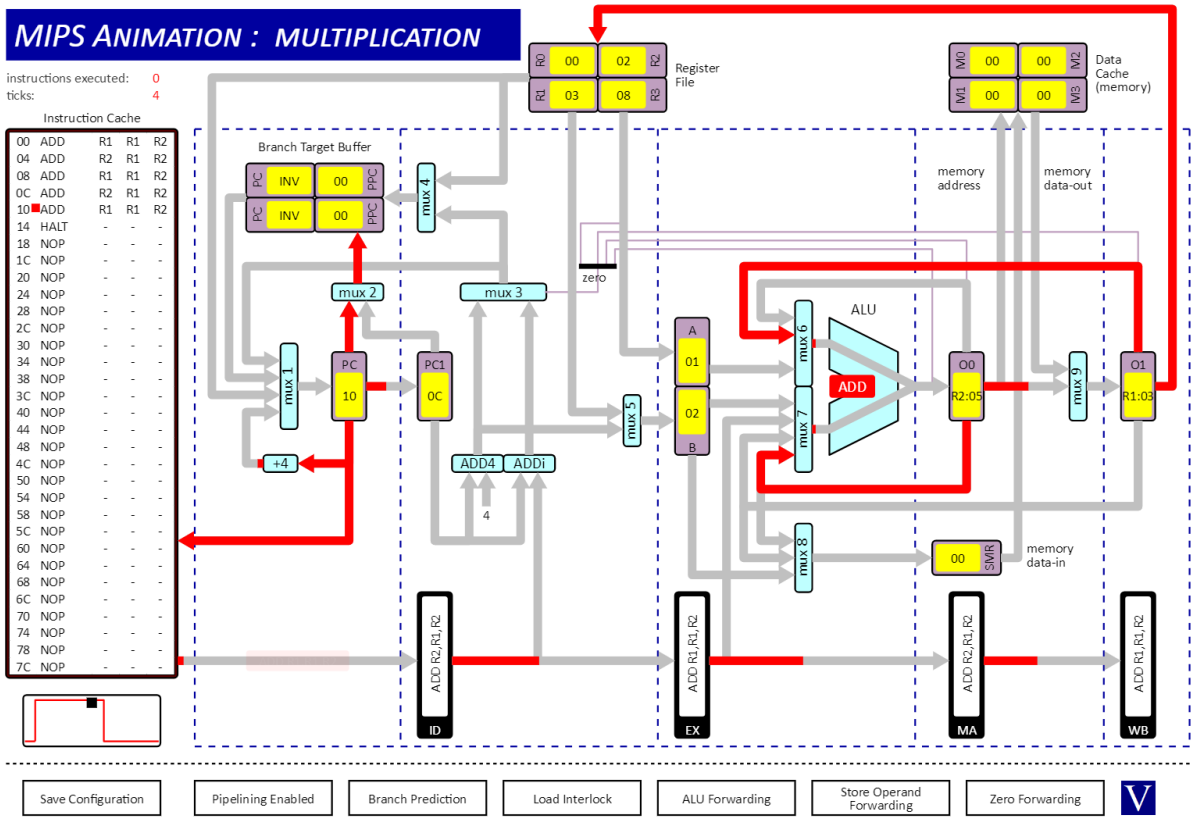
PART 1

ADD	R1,	R1,	R2
ADD	R2,	R1,	R2
ADD	R1,	R1,	R2
ADD	R2,	R1,	R2
ADD	R1,	R1,	R2



PART 2

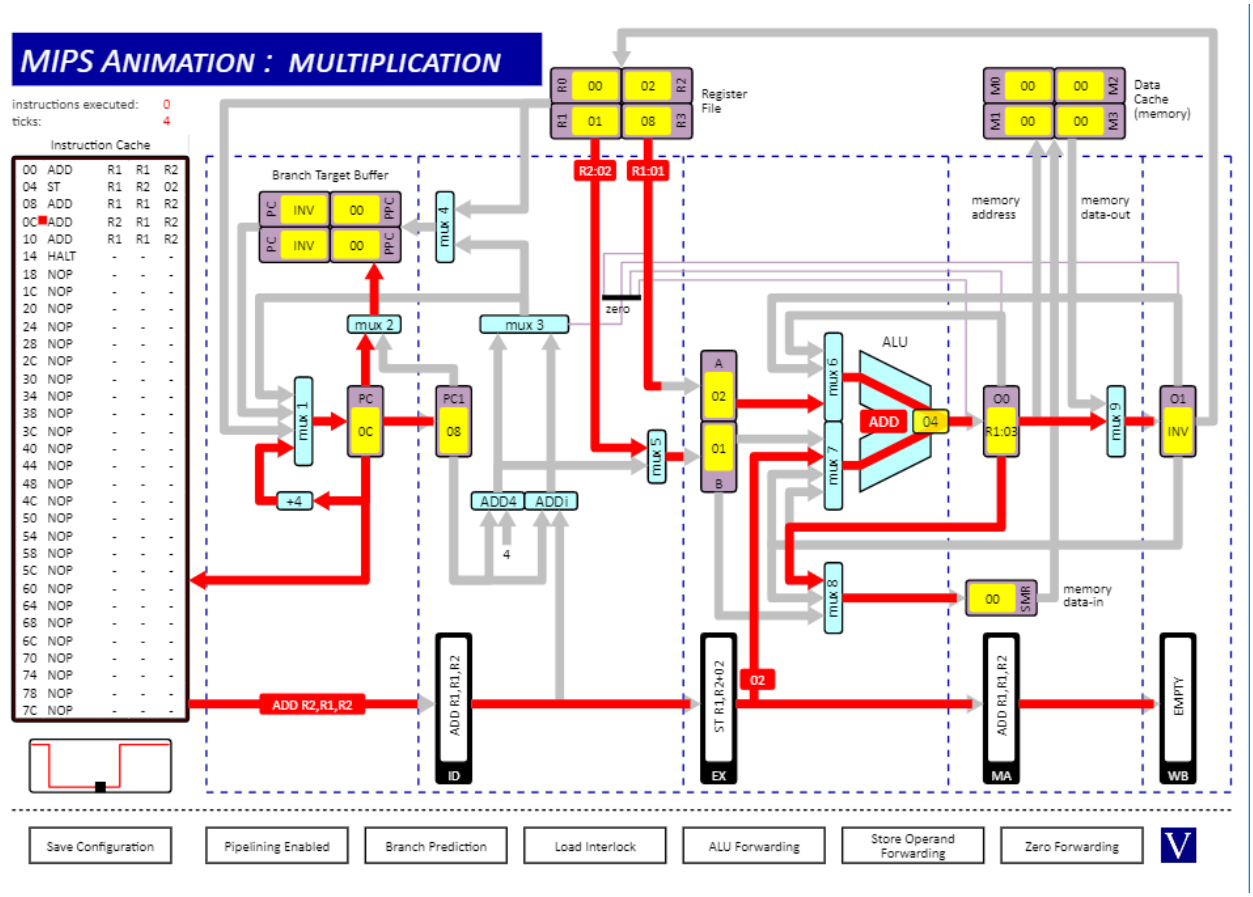
ADD	R1,	R1,	R2
ADD	R2,	R1,	R2
ADD	R1,	R1,	R2
ADD	R2,	R1,	R2
ADD	R1,	R1,	R2



PART 3

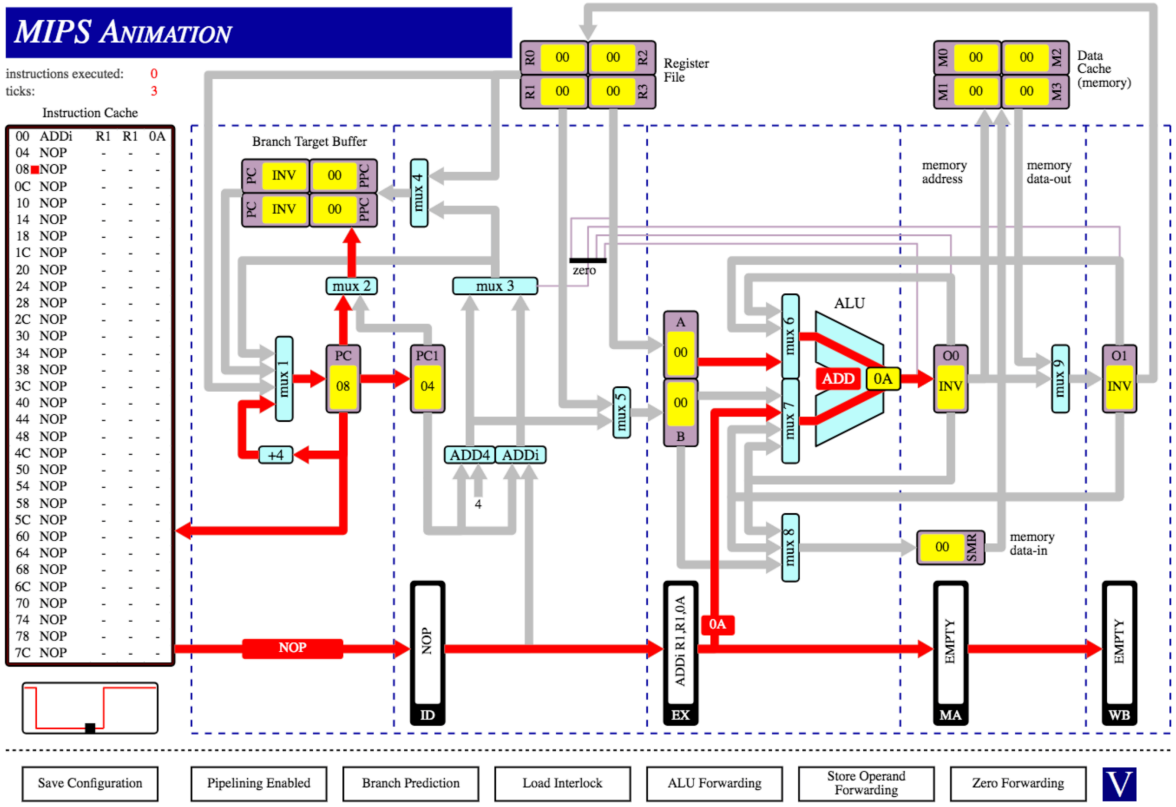
ADD R1, R1, R2

ST R1, R2, O2



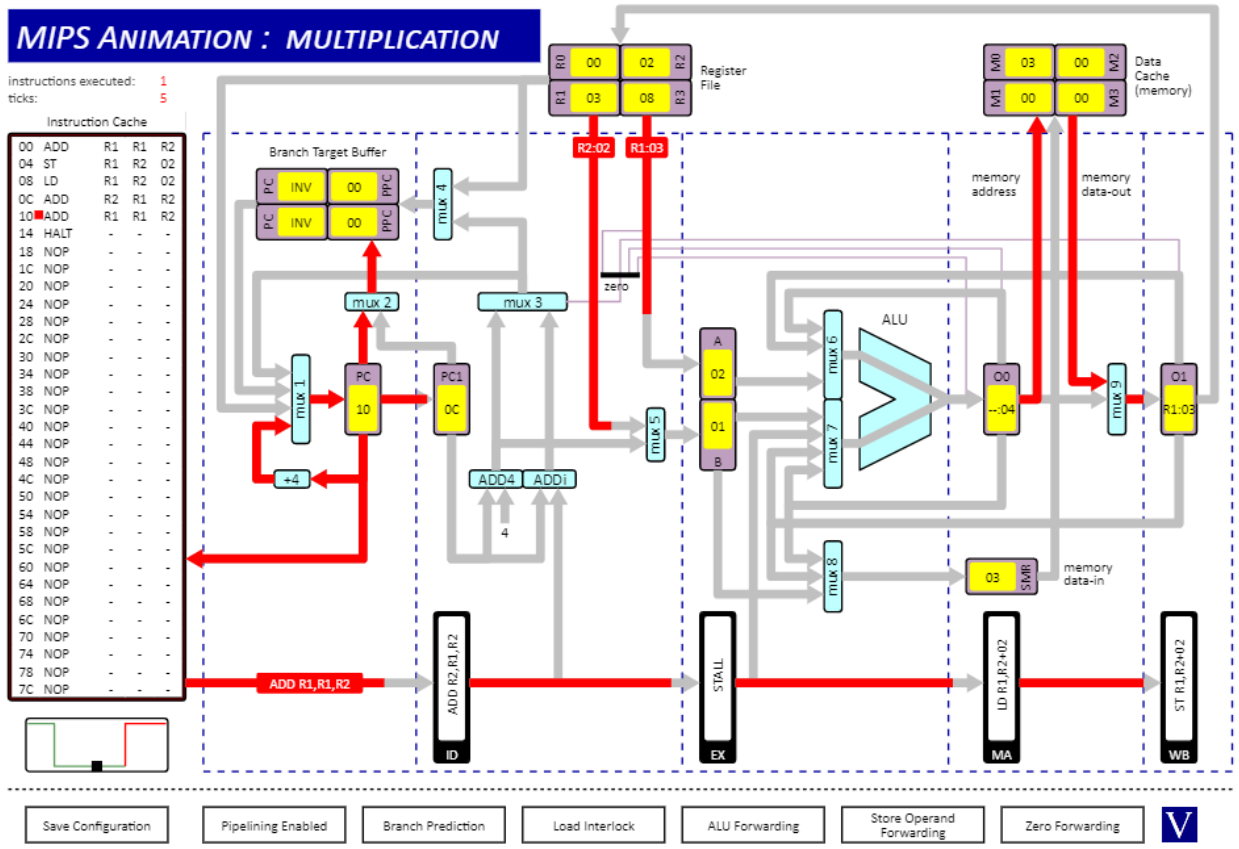
PART 4

ADDi R1, R1, OA



PART 5

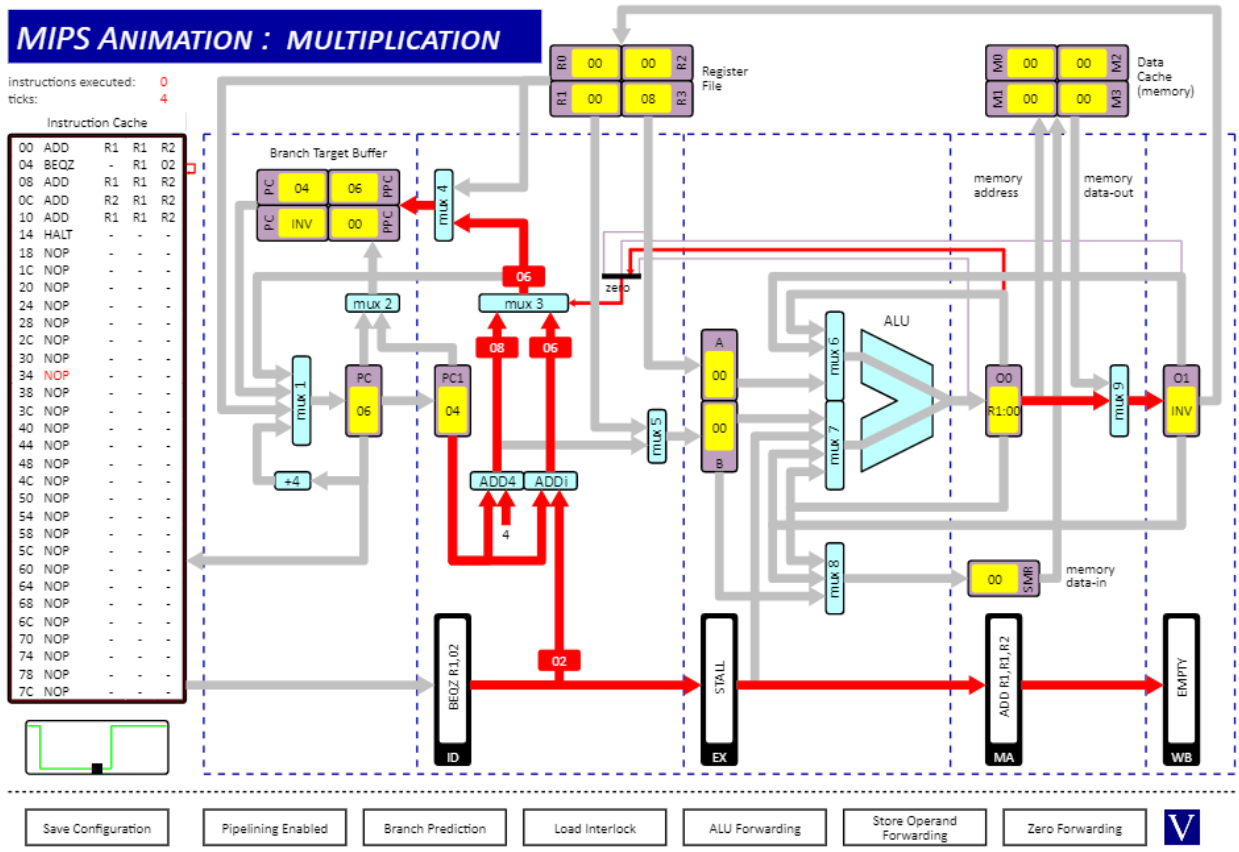
ADD	R1,	R1,	R2
ST	R1,	R2,	O2
LD	R1,	R1,	R2



PART 6

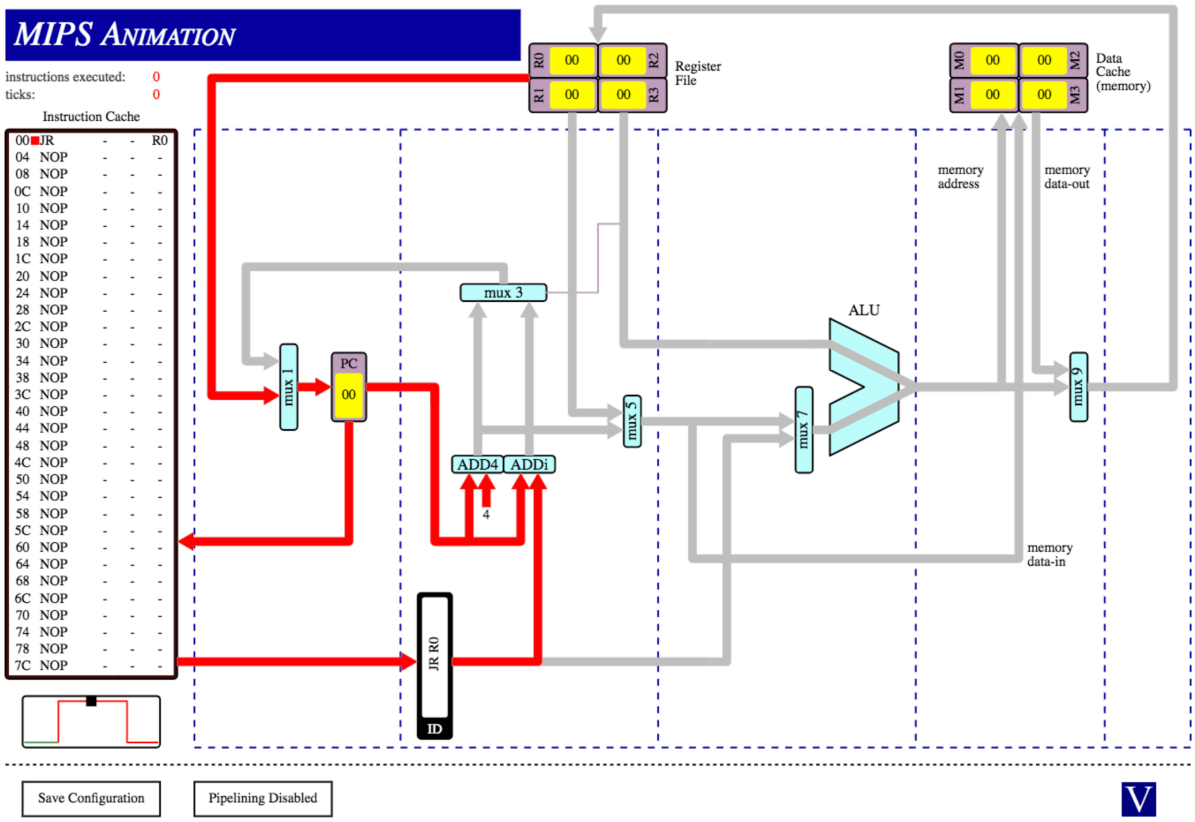
ADD R1, R1, R2

BEQZ -, R1, O2



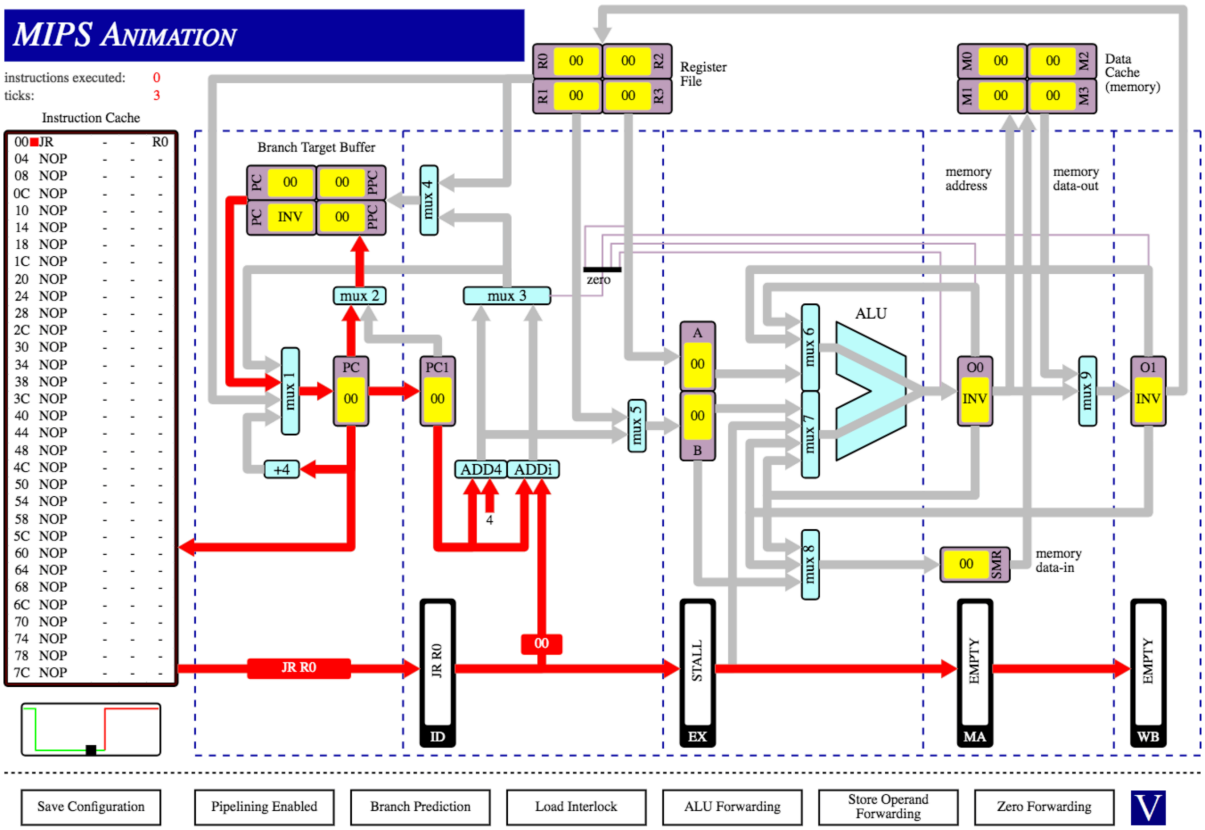
PART 7

JR -, -, R0


MIPS ANIMATION
instructions executed: 0
ticks: 0
Instruction Cache
00 JR - - R0
04 NOP - - -
08 NOP - - -
0C NOP - - -
10 NOP - - -
14 NOP - - -
18 NOP - - -
1C NOP - - -
20 NOP - - -
24 NOP - - -
28 NOP - - -
2C NOP - - -
30 NOP - - -
34 NOP - - -
38 NOP - - -
3C NOP - - -
40 NOP - - -
44 NOP - - -
48 NOP - - -
4C NOP - - -
50 NOP - - -
54 NOP - - -
58 NOP - - -
5C NOP - - -
60 NOP - - -
64 NOP - - -
68 NOP - - -
6C NOP - - -
70 NOP - - -
74 NOP - - -
78 NOP - - -
7C NOP - - -
R0 00 00 R2
R1 00 00 R3
Register File
M0 00 00 M2
M1 00 00 M3
Data Cache (memory)
memory address
memory data-out
memory data-in
mux 1
PC
00
mux 3
mux 5
mux 7
ALU
mux 9
ADD4
ADDi
4
JR R0
ID
Save Configuration
Pipelining Disabled

PART 8

JR -, -, R0


MIPS ANIMATION
instructions executed: 0
ticks: 3
Instruction Cache
00 JR - - R0
04 NOP - - -
08 NOP - - -
0C NOP - - -
10 NOP - - -
14 NOP - - -
18 NOP - - -
1C NOP - - -
20 NOP - - -
24 NOP - - -
28 NOP - - -
2C NOP - - -
30 NOP - - -
34 NOP - - -
38 NOP - - -
3C NOP - - -
40 NOP - - -
44 NOP - - -
48 NOP - - -
4C NOP - - -
50 NOP - - -
54 NOP - - -
58 NOP - - -
5C NOP - - -
60 NOP - - -
64 NOP - - -
68 NOP - - -
6C NOP - - -
70 NOP - - -
74 NOP - - -
78 NOP - - -
7C NOP - - -
Register File
Data Cache (memory)
Branch Target Buffer
mux 1
mux 2
mux 3
mux 4
mux 5
mux 6
mux 7
mux 8
mux 9
ALU
zero
memory address
memory data-out
memory data-in
+4
ADD4
ADDi
4
JR R0
IF
ID
EX
MA
WB
STALL
EMPTY
Save Configuration
Pipelining Enabled
Branch Prediction
Load Interlock
ALU Forwarding
Store Operand Forwarding
Zero Forwarding

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

i) Instructions Executed: 22
Cycles: 28

The reason there are more cycles completed then instructions executed is due to the branching in the program. When the program branches or jumps, stall cycles occur. These are cycles where one stage in the pipeline is not in use meaning it cannot contribute to the execution of instructions. This then leads to more cycles in order to execute a given instruction. Then at the end of the program, seeing as the pipeline executes 5 different stages of the instruction, the full pipeline needs to be executed in order for the final instruction to be executed. This means it adds the extra cycles needed for the final instruction execution.

ii) Instructions Executed: 22
Cycles: 29

Here we have extra cycles for very similar reasons above. In fact they are the same reasons as above, except here we have an extra cycle comparing to the cycles used above. This extra cycle comes from our branch prediction not being present. Before the branch prediction predicted the correct branch instruction however now that the branch prediction is not present, it must be calculated before branch.

iii) If we were to swap the two shift instructions we would have to wait an extra clock cycle. This is due to the LD instruction beforehand, we must wait until the value has been brought into the register before we can then use it in our shift instruction.