****

CSCE 330

Computer Architecture

Dr. Khaled El Ayat

MIPS Pipelined Processor  
Project Report

Submitted by

Mohamed A. Tawfik

Mohamed S. El Minawi

Ahmed Refaay

Mohamed Y. Ali

Outline

* Stages
* Jumps, Branching, stalls and forwarding
* Test Cases

**Stages and buffers**

* PC, aka IF
* IS buffer
* IS (Instruction Second)
* Register File buffer
* RF (Register File)
* Execution Buffer
* EX
* Data fetch buffer
* DF (Data Fetch)
* DS buffer
* DS (Data Second)
* TC buffer
* TC (Check tag in the cache but assumed that hit is present in all)
* WB buffer
* WB

**Jumps, Branching, stalls and forwarding**

We aim to detect all “jump” instruction during the Register file stage, hence causing a constant and definite penalty of 2 cycles.

Regarding “Branch” instructions; a class “Branch predictor” is created and it aims to predict where the Branching would occur in the execution stage. Due to the practicality of such a class there are two possibilities:   
**Best case** where no stalling occurs and the right prediction in execute stage was taken.

**Worst case** where the prediction was wrong in EX, hence we pay a penalty of 3 cycles. PC will be depending on that, hence we flush the IF, RF & EX and we flush the buffer before the execution stage.

“Load” instruction would have 3 possible stalls.

Stall for load is detected in execute stage, if there is a problem, we store PC and IS IF & EX buffers.

We flush Data Fetch buffer, hence will lead to a max penalty for load stall as 2, 1 cycle if there was no need and 0 if no stalls were required.  
  
Forwarding unit: It forwards data in the EX stage from all the possible sources IF, IS and TC, and if need from WB, we write before reading so there won’t be a problem.

**Cases**

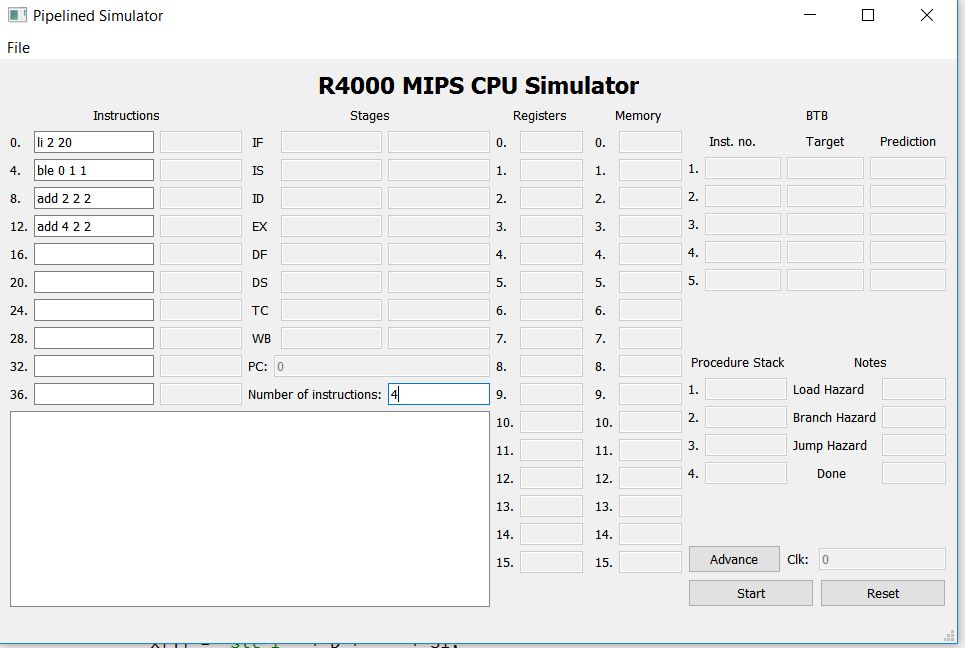
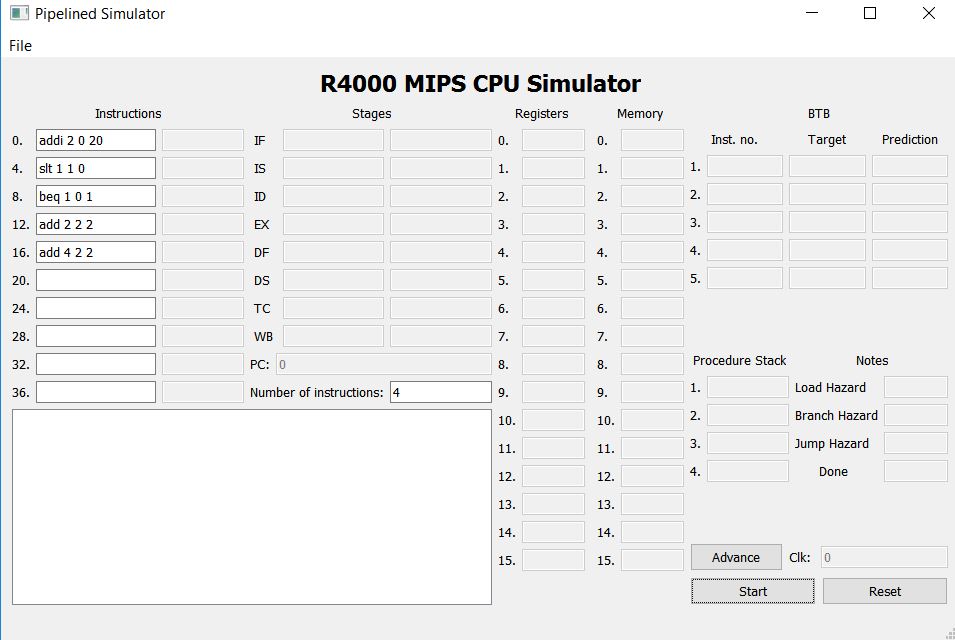
**First Test case:**

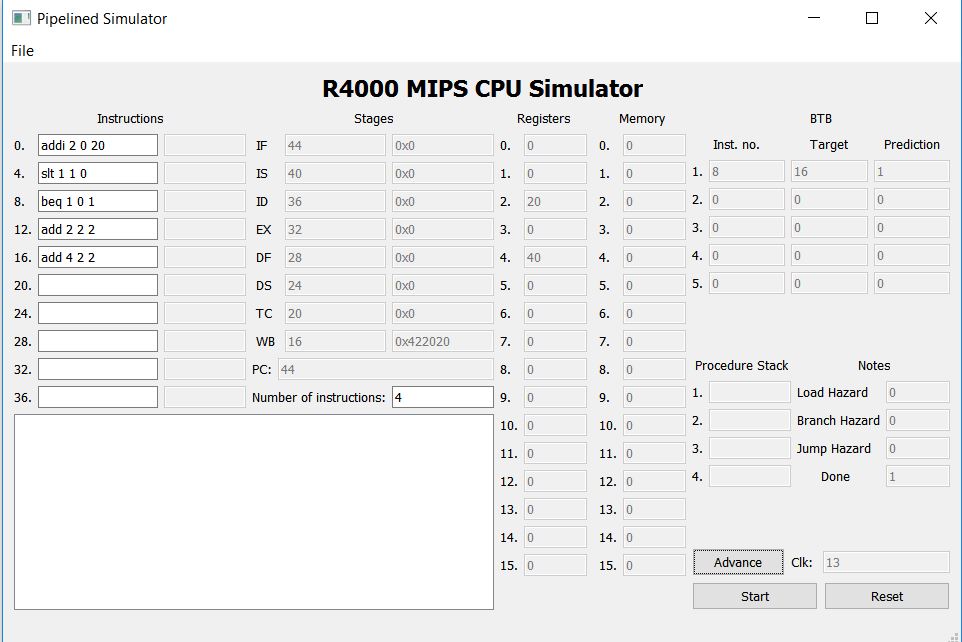
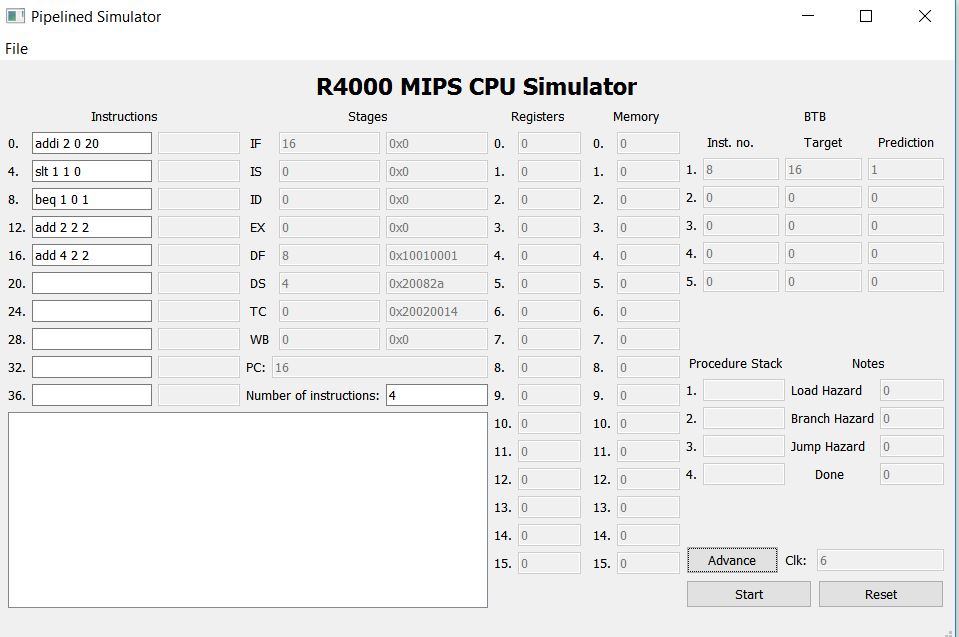
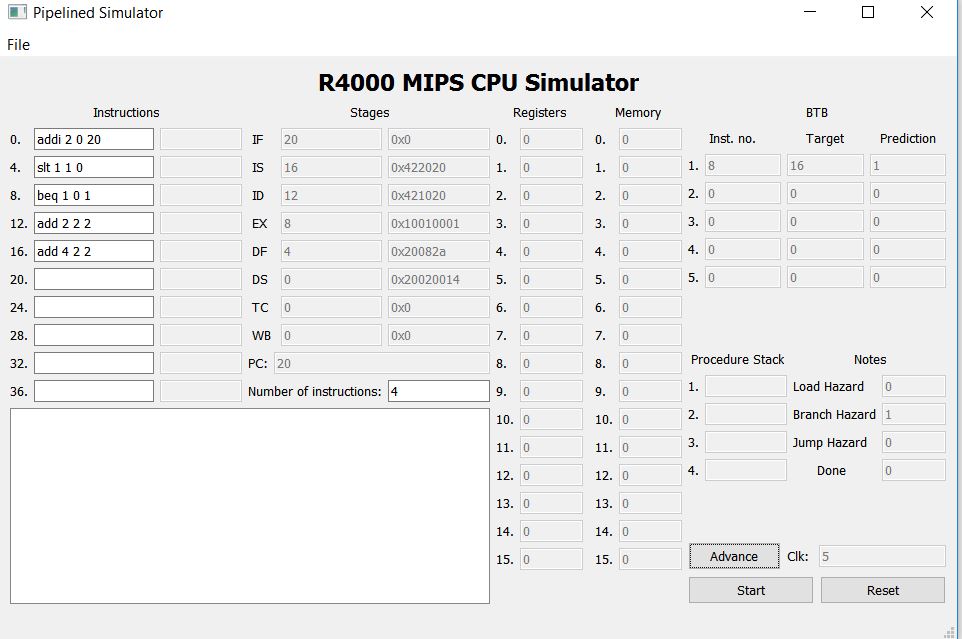
addi 2 0 20

ble 1 1 1

add 2 2 2

add 4 2 2

This test case is mainly target to asses the effectiveness of the pseudo to true instrcutions, in this case “ble” instruction. It is clear that it has been converted to slt and beq.  
  
Screenshot: ****

****

**Second Test Case:**

li 2 20

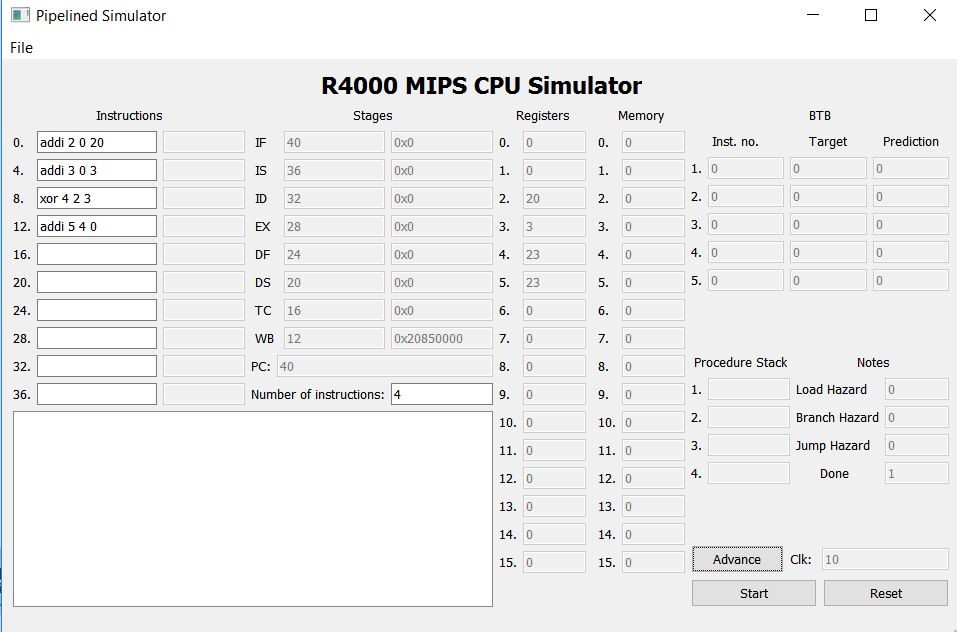
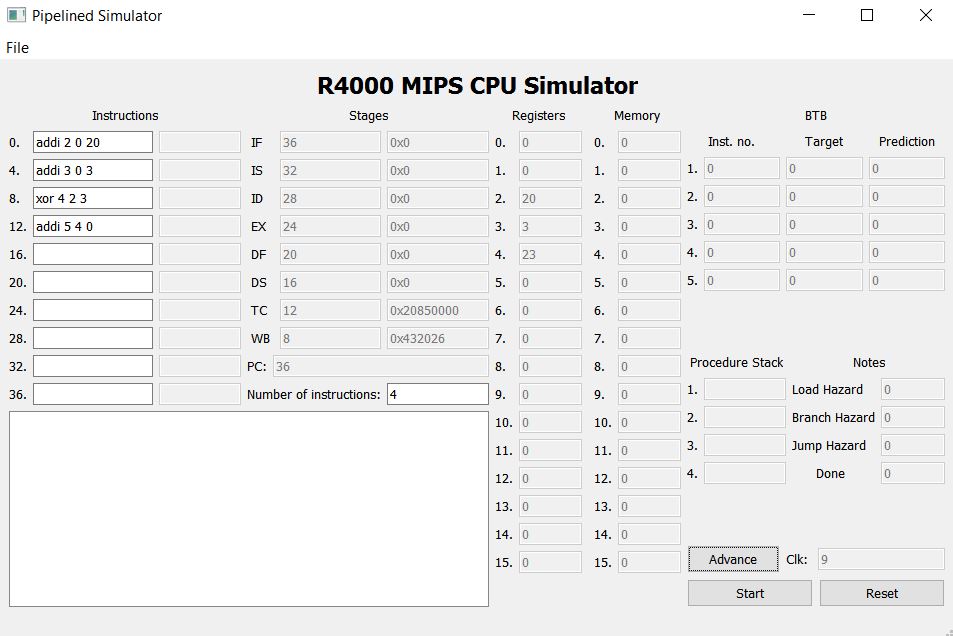
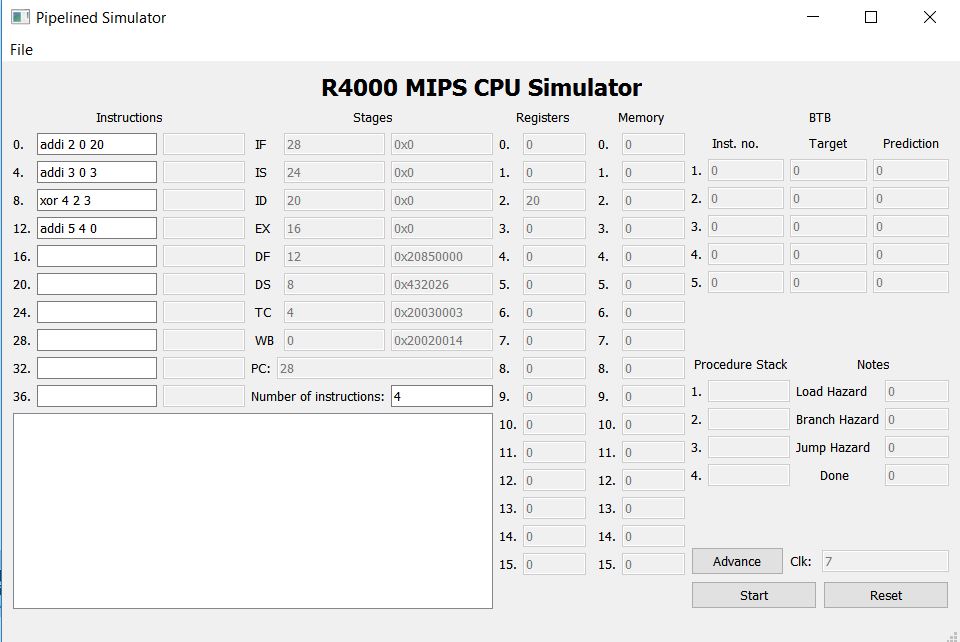
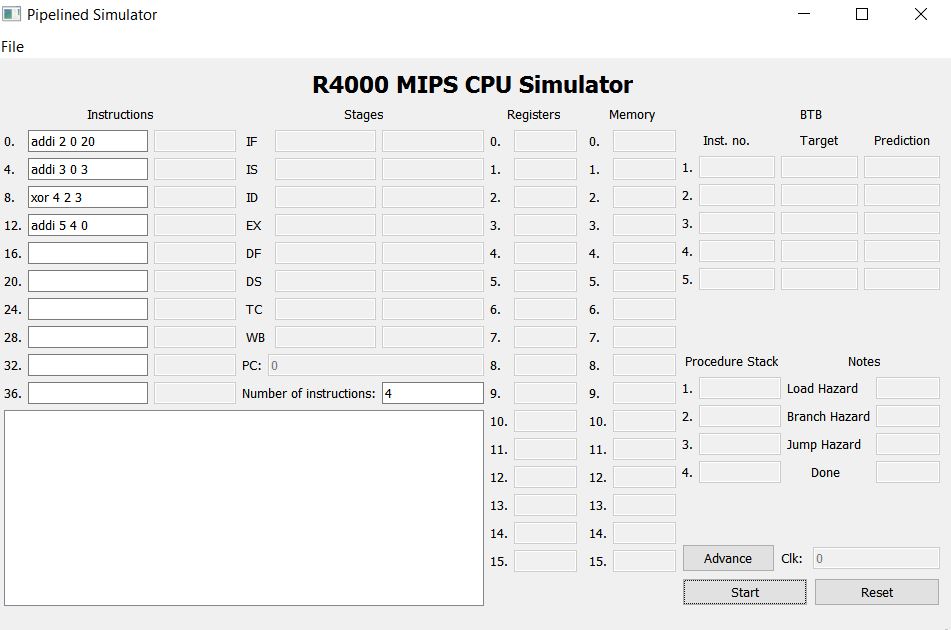
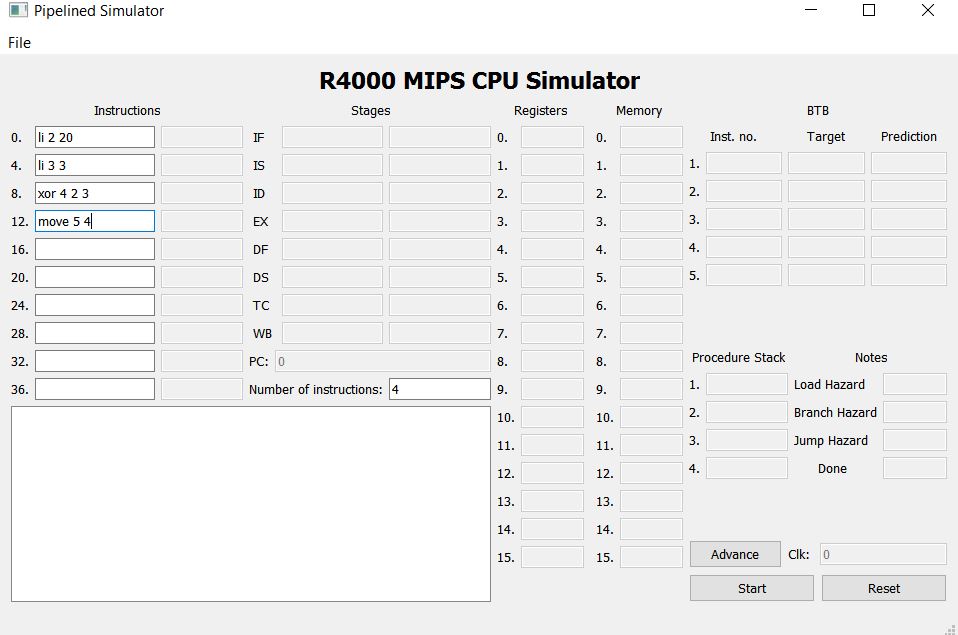
li 3 3

xor 4 2 3

move 5 4

This case is aimed to show the forwarding, as there is a data dependency xor and li, then between move and xor.

Screenshots:

****

**Test Case 3:**

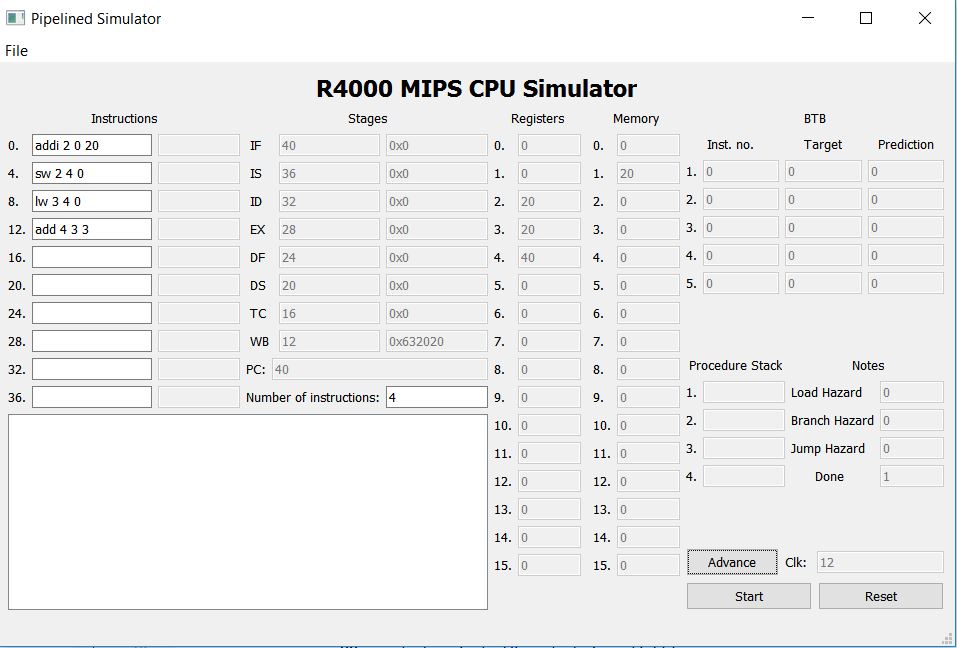
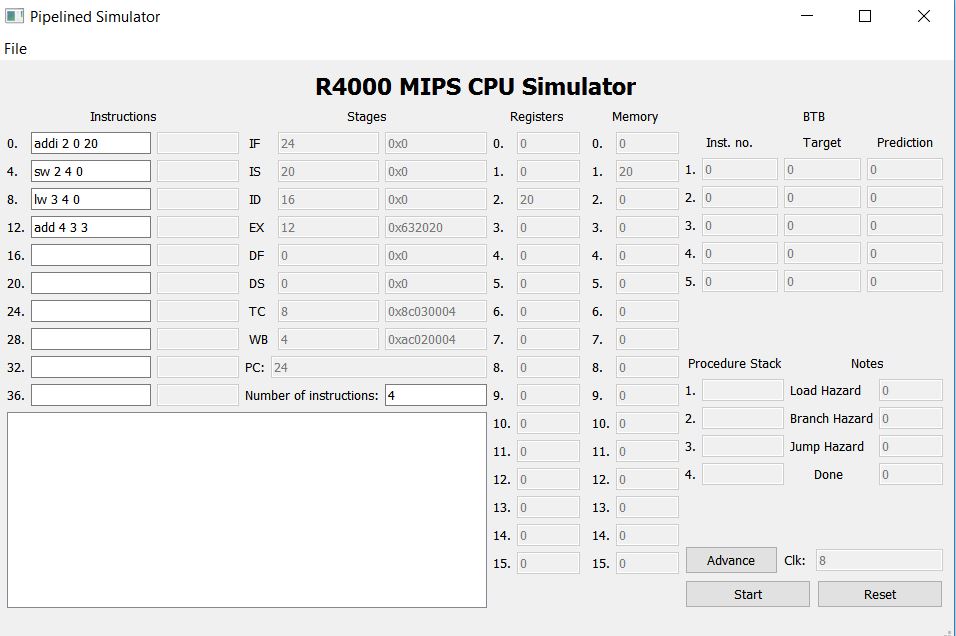
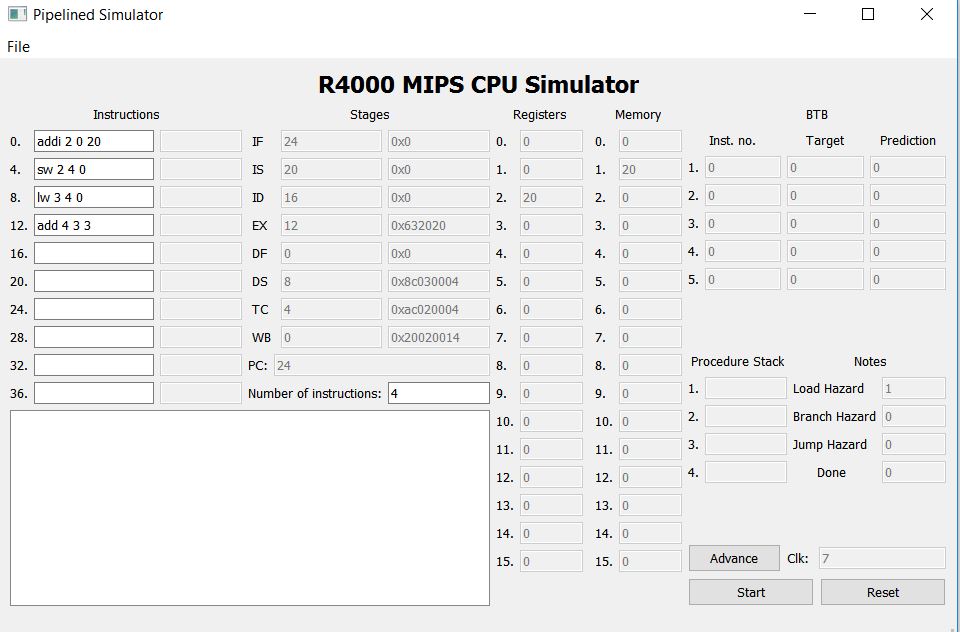
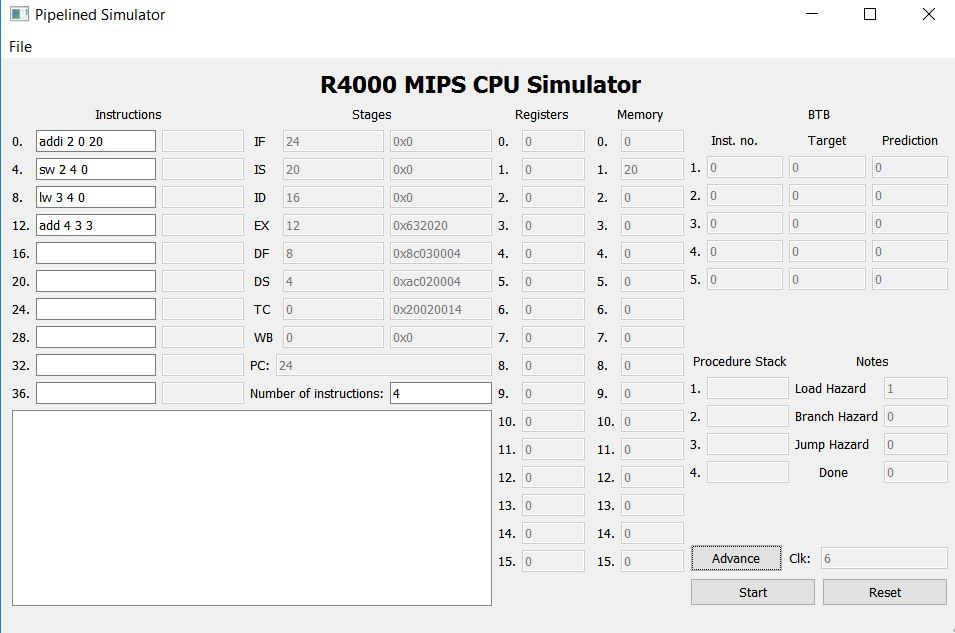
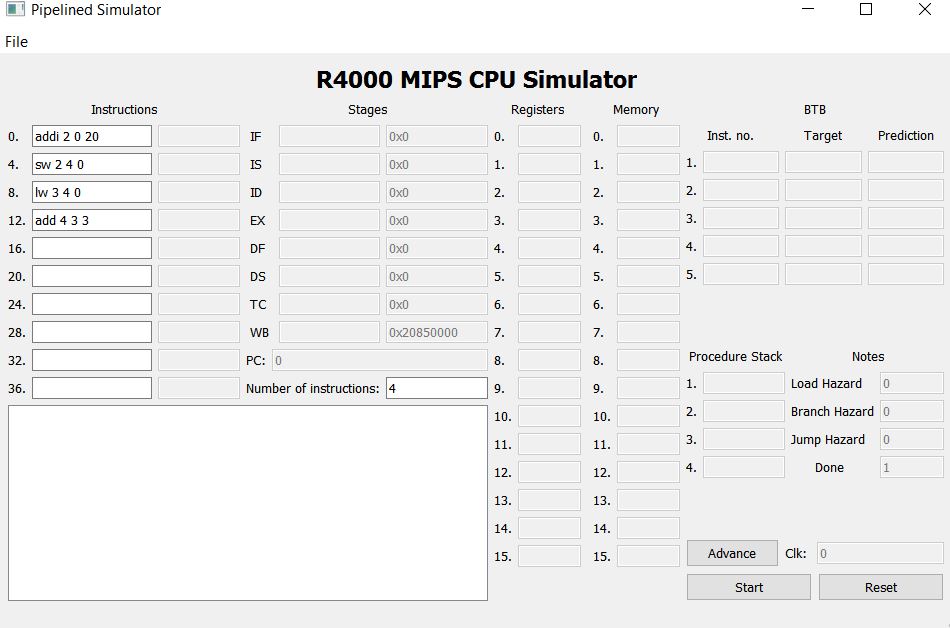
addi 2 0 20

sw 2 4 0

lw 3 4 0

addi 4 3 3

This case is aimed to show the storing in the memory (sw) and loading from the Memory to registers (lw), it also tackles the stall by the hazard unit.

Screenshots: ****

**Test Case 4:**

jpr 2

li 5 15

j 4

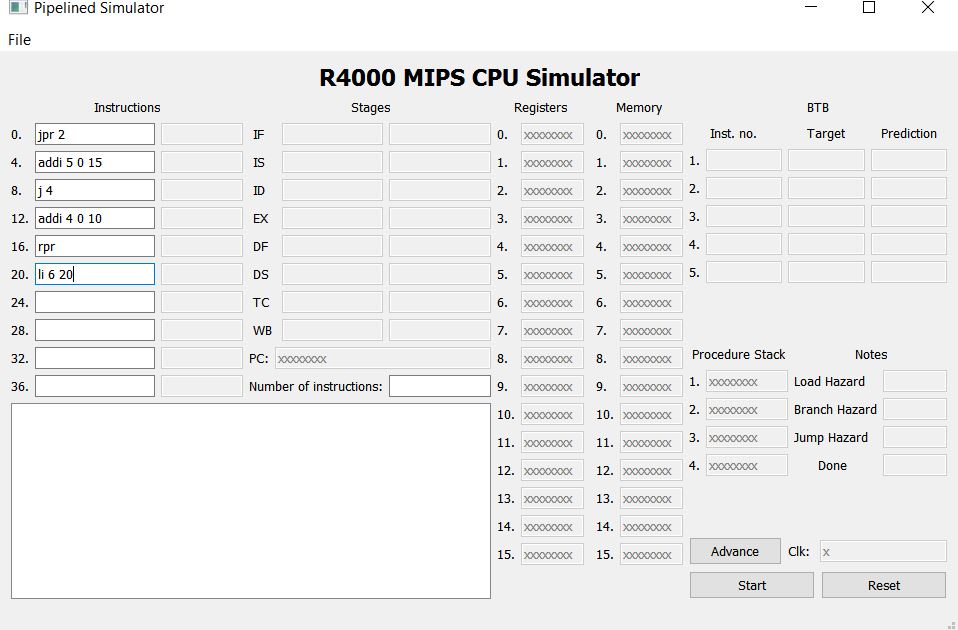
addi 4 0 10

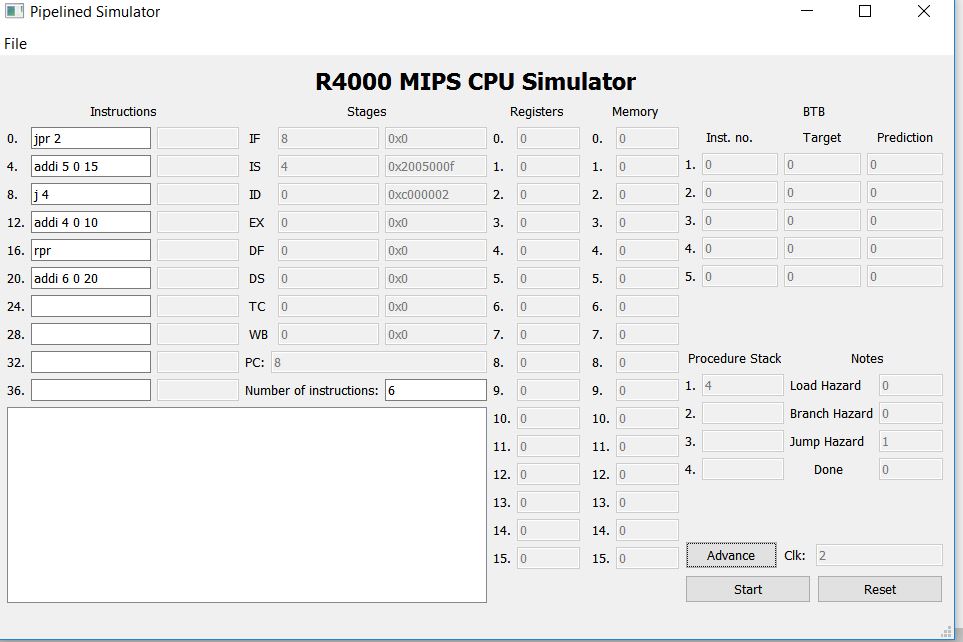
rpr

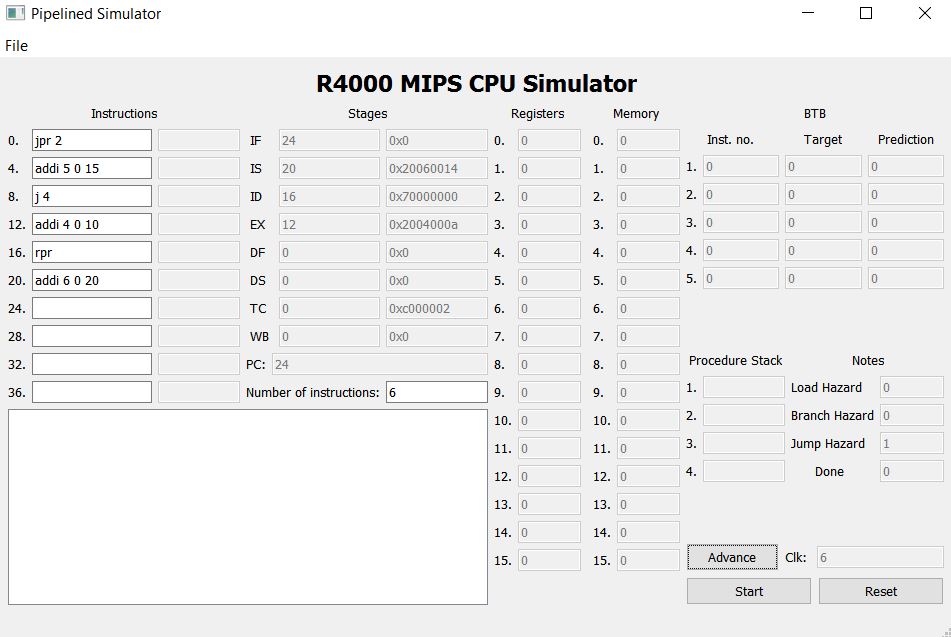
li 6 20

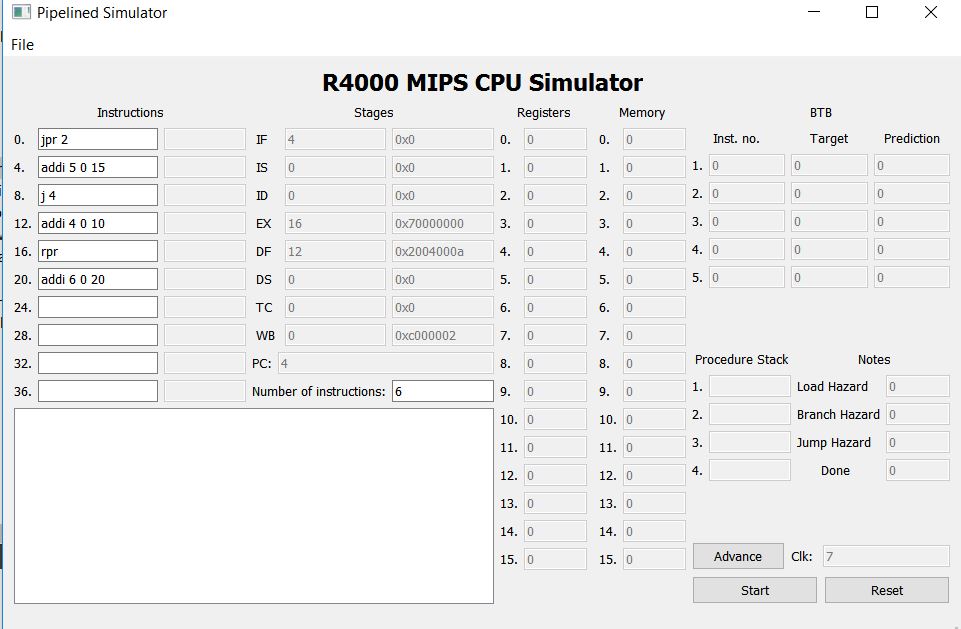
This case shows the jump procedure and detecting the jump hazard.

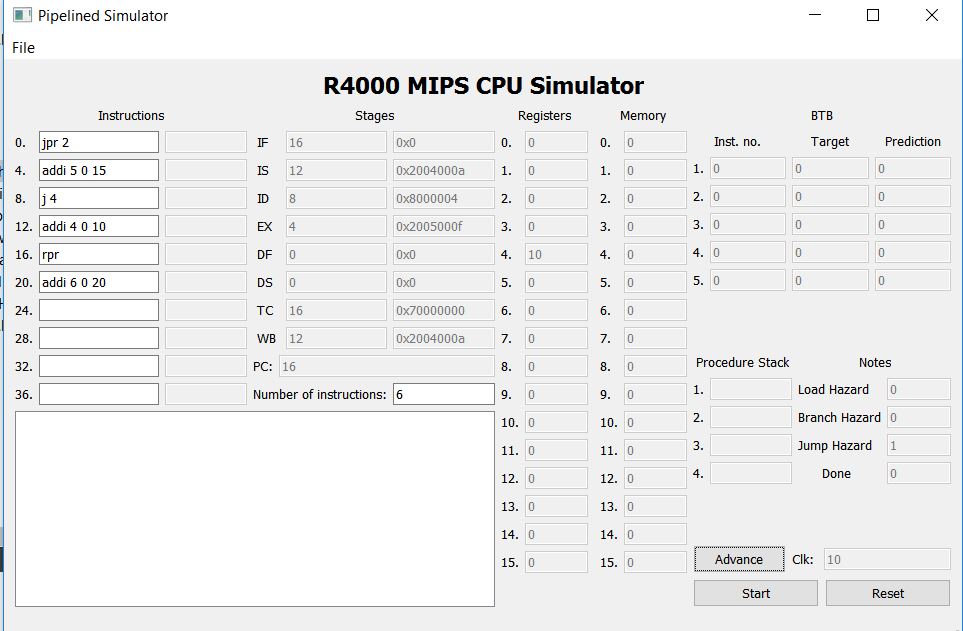
**Screenshots:**

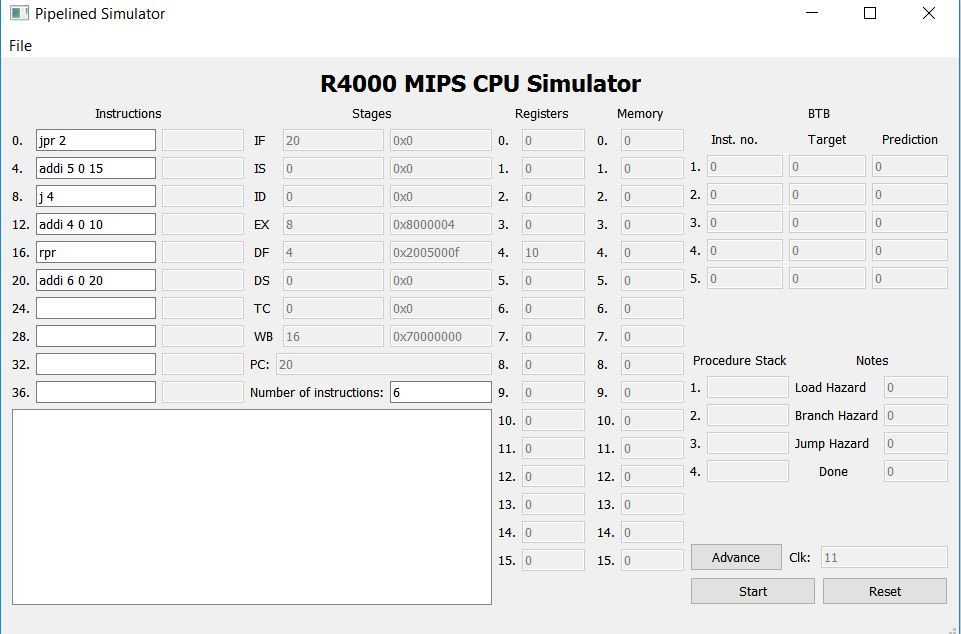


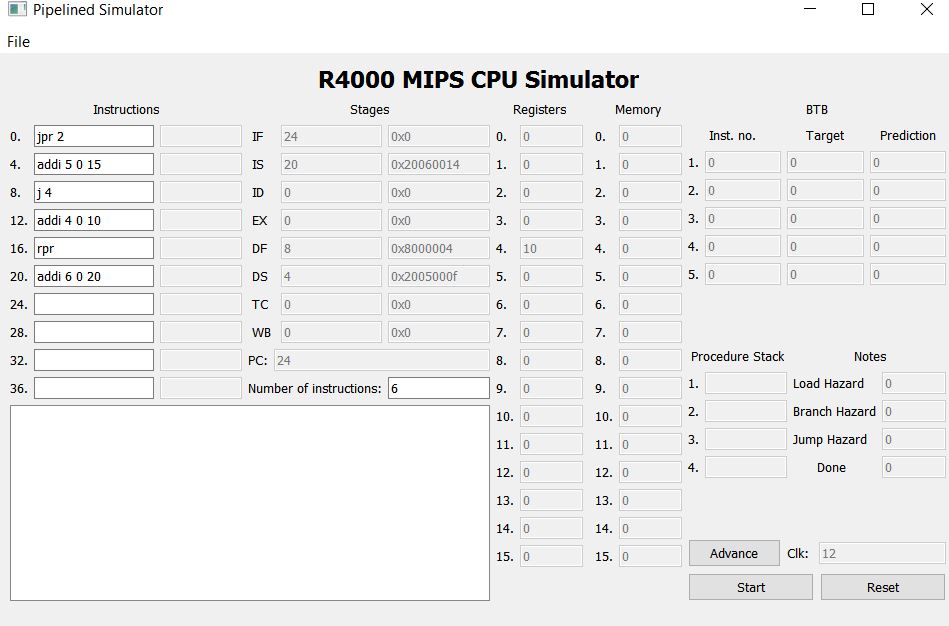


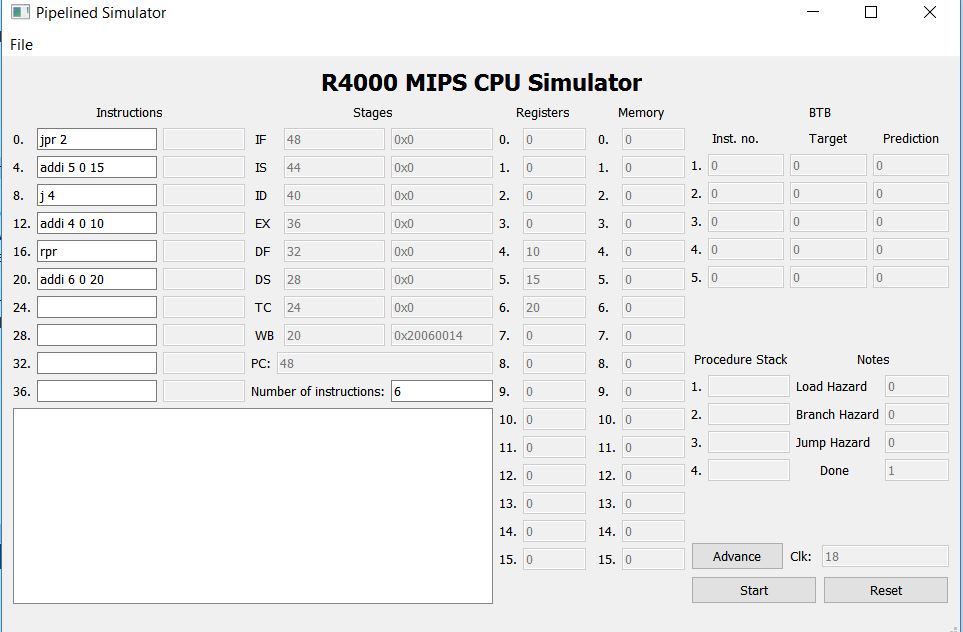












**Test Case 5:**

li 2 -10

li 3 20

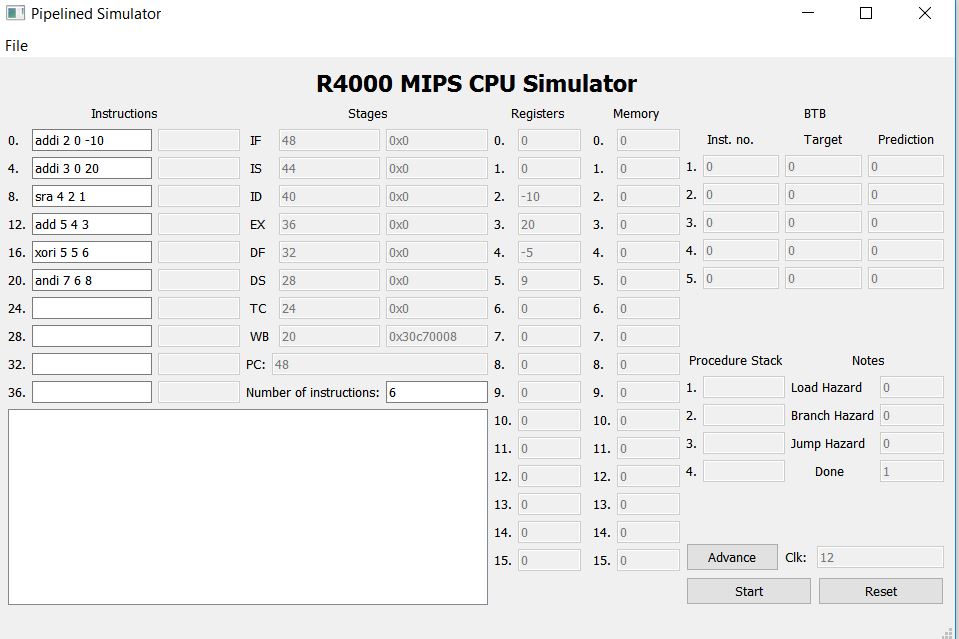
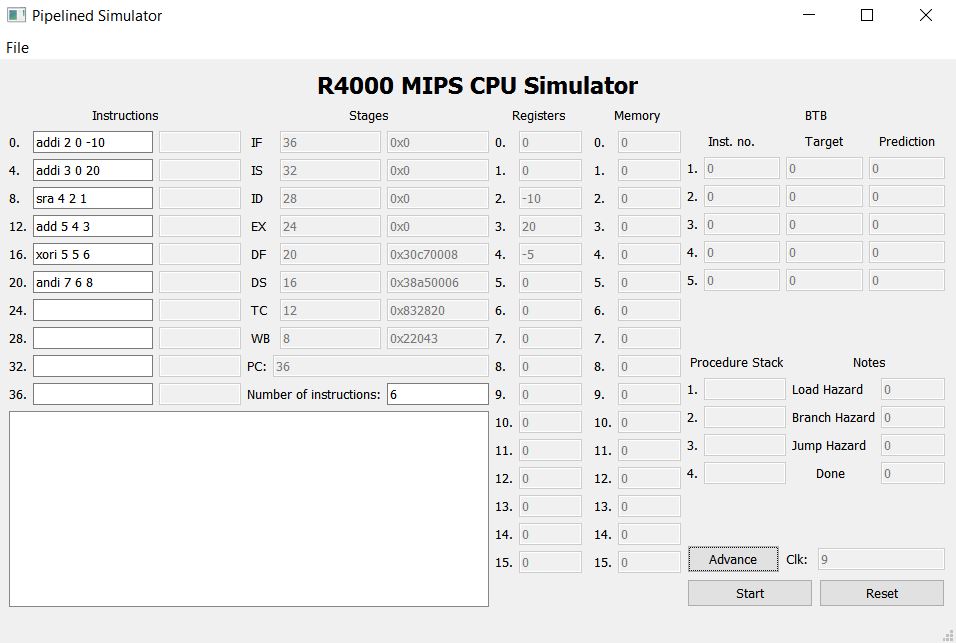
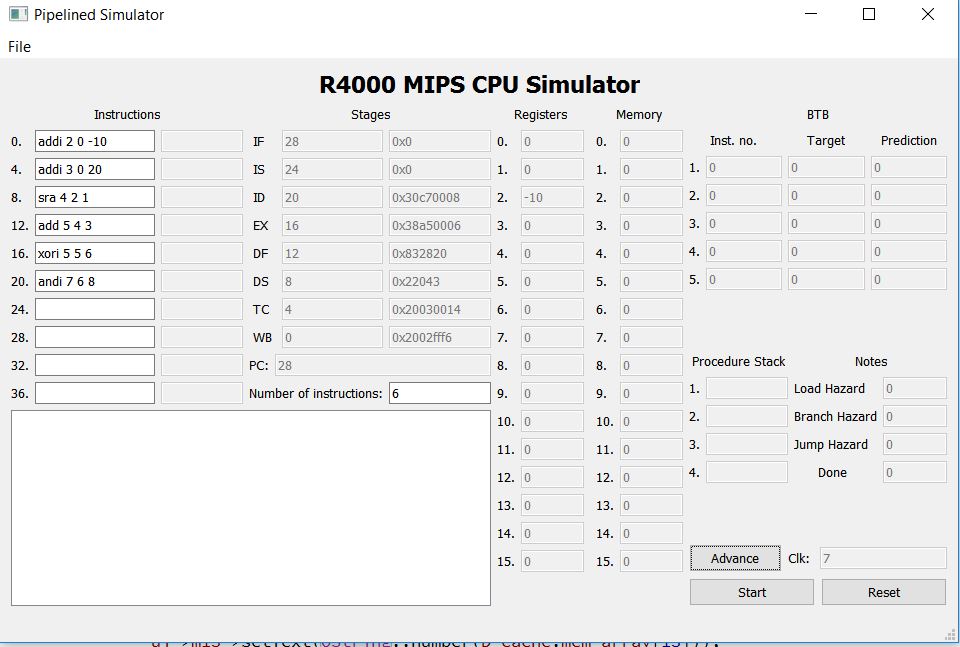
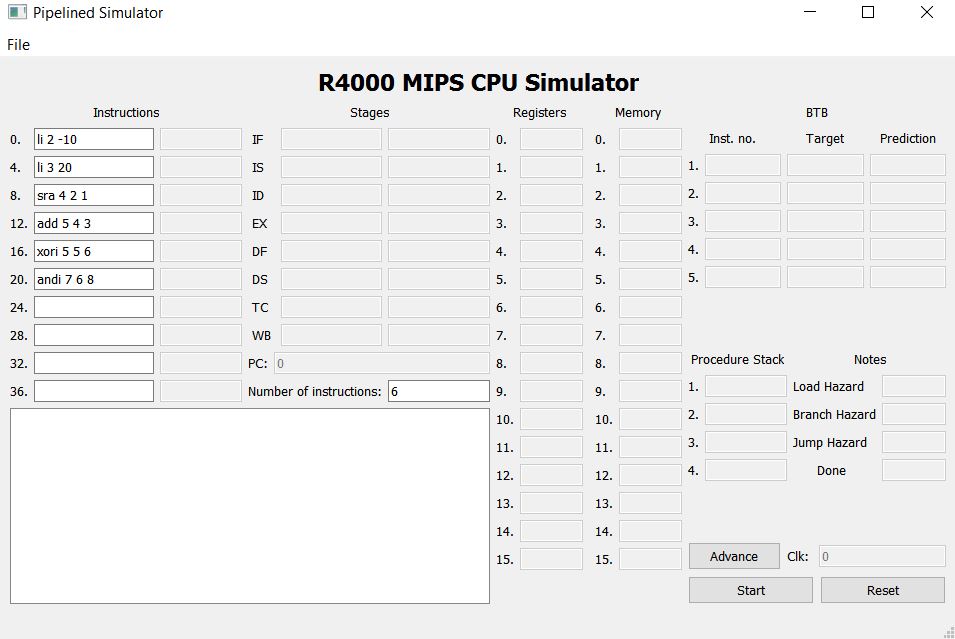
sra 4 2 1

add 5 4 3

xori 5 5 6

andi 7 6 8

This case shows other instructions to show that we support almost every MIPS instruction.

**Screenshots:**

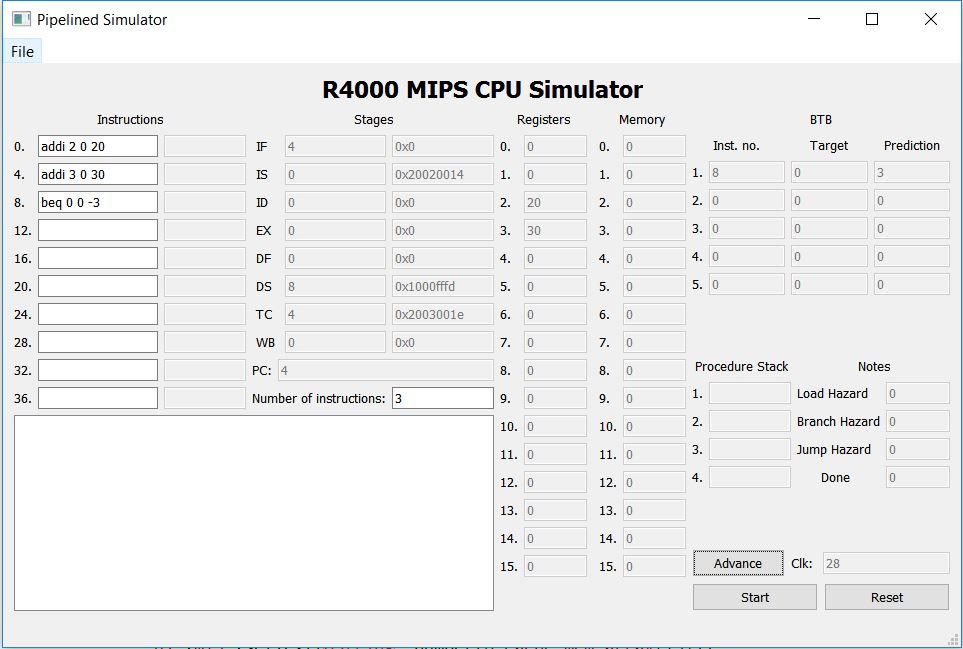
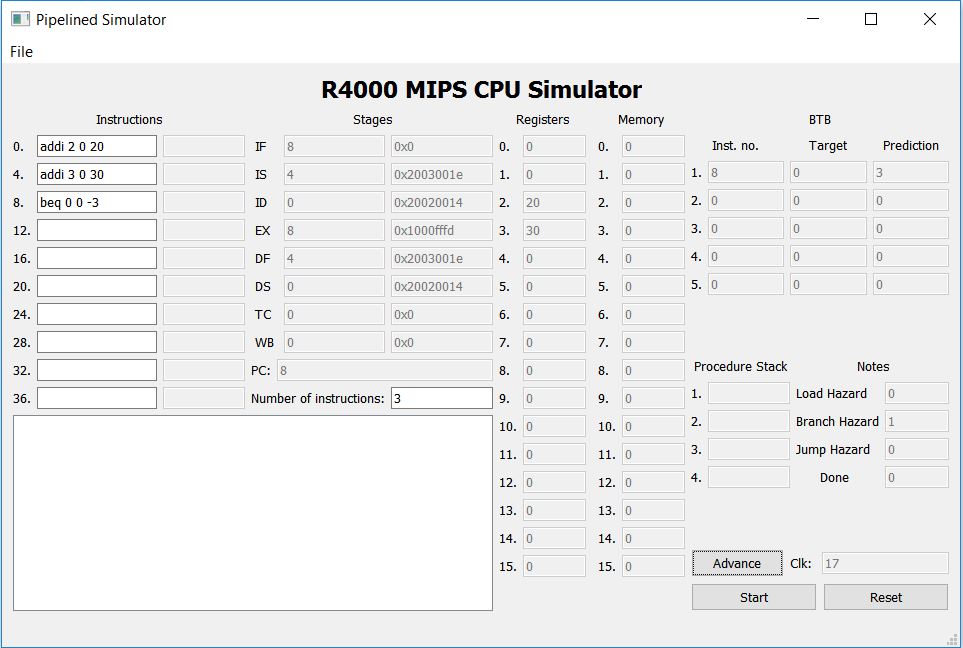
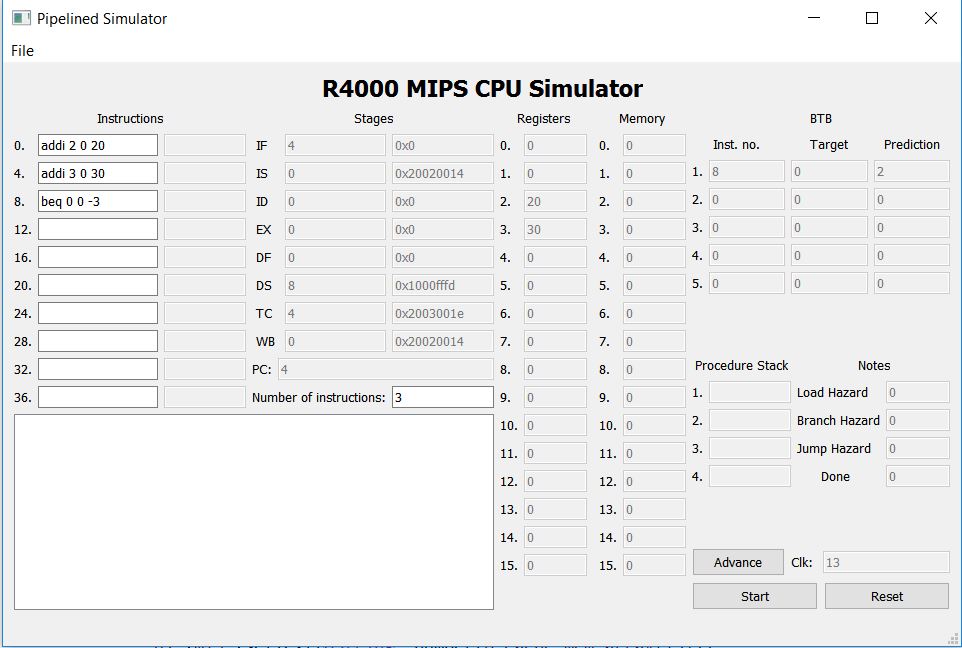
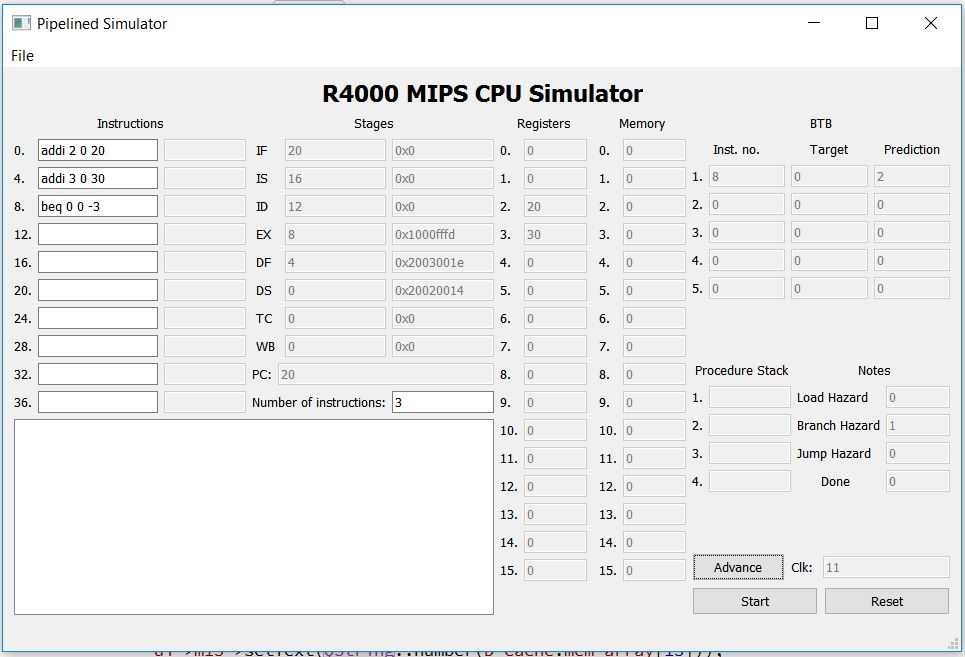
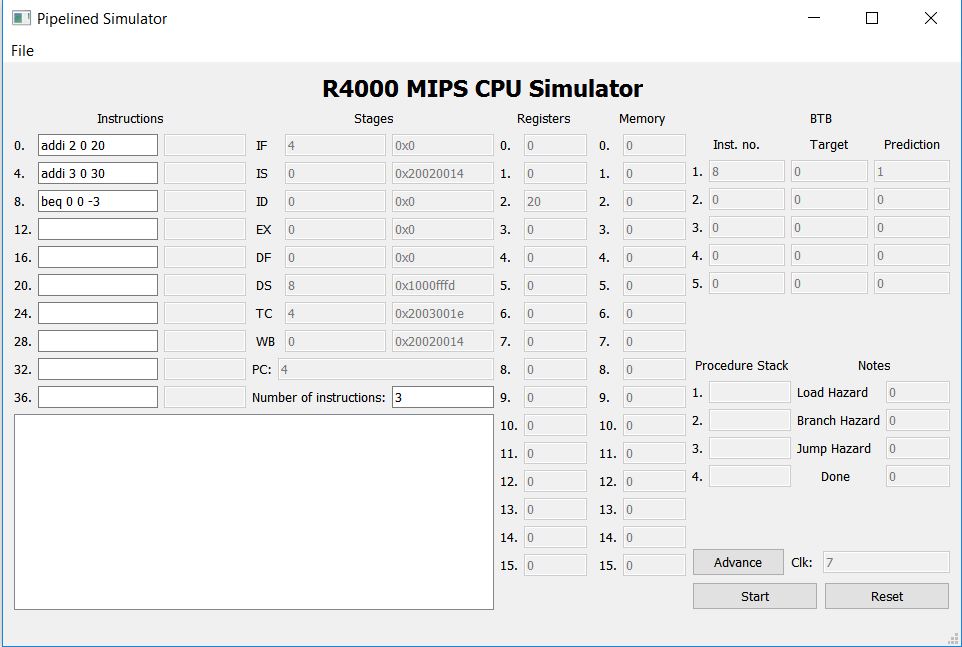
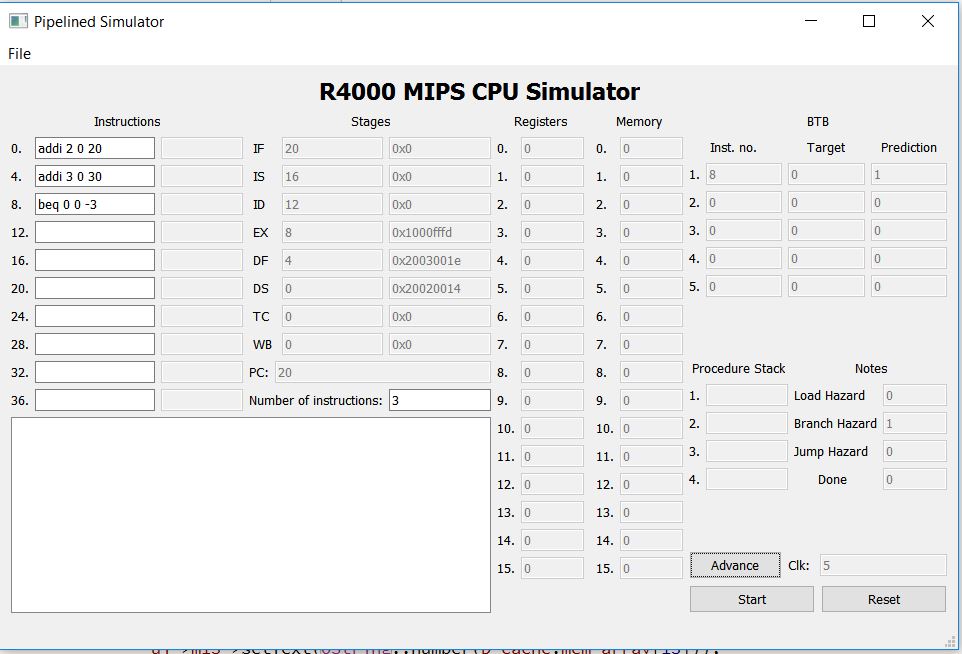
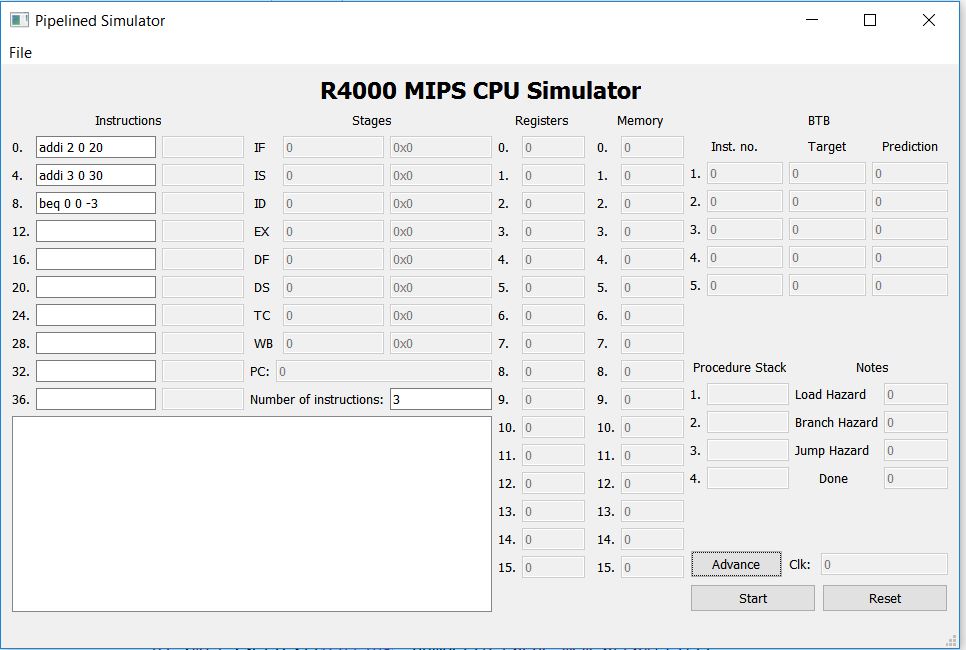
**Test Case 6: (Infinte loop)**

addi 2 0 20

addi 3 0 30

beq 0 0 -3

This is our Branching case, it is an infinite case to show how are BTB is updates with each one. We specified the 2 bits as 0 and 1 for not taken, 2 and 3 for taken.

**Screenshots:**