CSCE 3301 - computer Architecture

Project (1): RV32IC implementation and testing

Fall 2019

Submitted by:

Mariam Abul-Ela	900141674
Nada Badawy	900171975
Mohamed Al-Awadly	900163100

Instructor: Dr. Sherif Salama

Date: 11/11/2019

Abstraction:

In this milestone we used the single cycle data-path we did in the previous milestone and added the pipelined register to it. moreover, we were asked to modify the data-path to have only one memory for both instructions and data. By re-implementing the data-path with one memory. we re-implement this data-path in three stages as follow

Unified memory:

In order to make one memory for both data and instruction we made many changes, first we generate a new slower clock by using T-flip. we use the new clock and alternate between its negative and positive edge. For the PC, ID_EX, MEM_WB, we give them a positive age clock. While for the IF_ID, EX_MEM, and the register file works at the negative age of the new clock. This alternation between the edges make the pipelined as three stages, which fitch an instruction every two-clock cycle. Also, this simplifies the forwarding logic as the only possible RAW data hazard can occur because of data dependency between two adjacent instructions and allows the handling of load-use hazards as normal RAW hazards that can be resolved using forwarding.

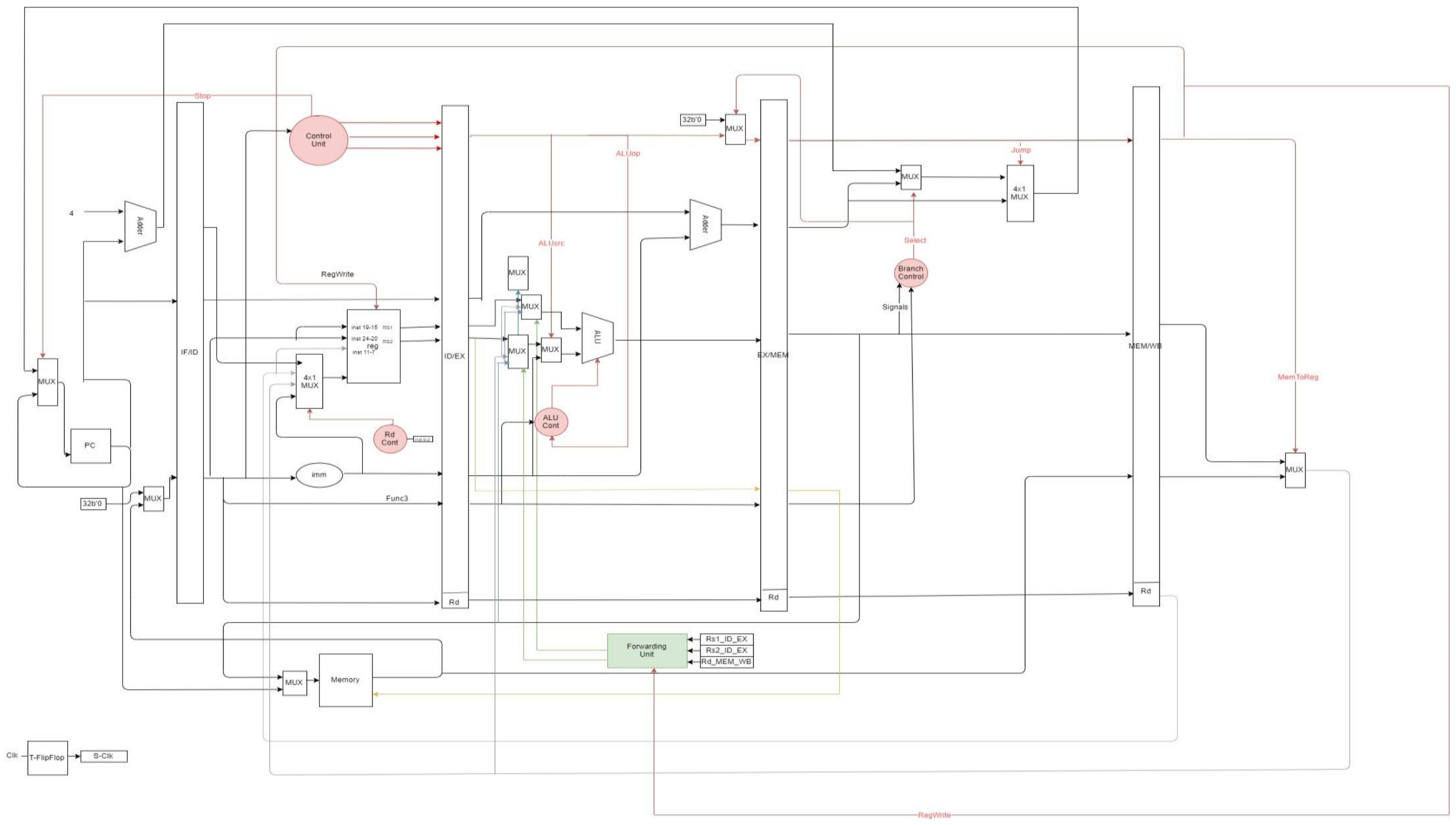
Forwarding unit and Hazzard:

By this implementation we encounter only two types of hazard.

Data Hazard: this happened when the two instructions are adjacent to each other and can be solved by the normal forwarding unit. Moreover, the load-used dependency now will not have to stall for one cycle and can be solved by the normal forwarding unit.

Control Hazard: this happened mainly because of the branch. We assume that the pipeline will have a static branch predictor of “Not Taken”, which means that the pipeline will fetching the following instructions of the branch whatever it is taken or not. When the branch is not taken this will not affect our results, however, when the branch is taken, there will be two instructions in the pipelined which are wrongly taken and needs to be flushed from the pipelined. In order to do so, we made a two 2X1 multiplexers to choose from 32 bits zeros or the real instruction. In case the branch is taken we will have the select signal equal to one, so we used this select signal to be the selection of the multiplexers. When the select is one it chooses the 32 zeros to act like a no-op instruction.

Implementation:



1. We took the implementation of the single cycle which we implemented in the milestone2.

2. We made the data path for the pipeline.

3. We implemented the registers (IF/ID register, ID/EX register, EX/MEM register, MEM/WB register) which will save the values to be used in the pipelining.

• IF/ID (96 bits) register will take the values of PC +4, PC counter and the instruction.

• ID/EX (157 bits) register will take the values of PC counter, PC +4, control unit output, rData1, rData2, the instruction, immediate generator.

• EX/MEM (115 bits) register will take the values of PC counter, PC+ immediate, ALU output, control unit signals, RD, immediate, rData2, func3, opcode

• MEM/WB (73 bits) register will take the values of control unit signals, Memory output, ALU output, RD.

4. We made a T Flip flop in order to make the instructions issue every two cycles and applied it to all register alternatively between the positive and negative age of the clock.

5. We added the forwarding unit which implemented in the lab by some modification which make it simpler as we are issuing the instruction every two cycles. This forwarding unit decides whether to forward data to the following instruction or not which helps to avoid data hazards.

6. We used the output of the branch control unit to detect whether the instruction is branching or not and accordingly the program decides whether to apply the flushing or not.

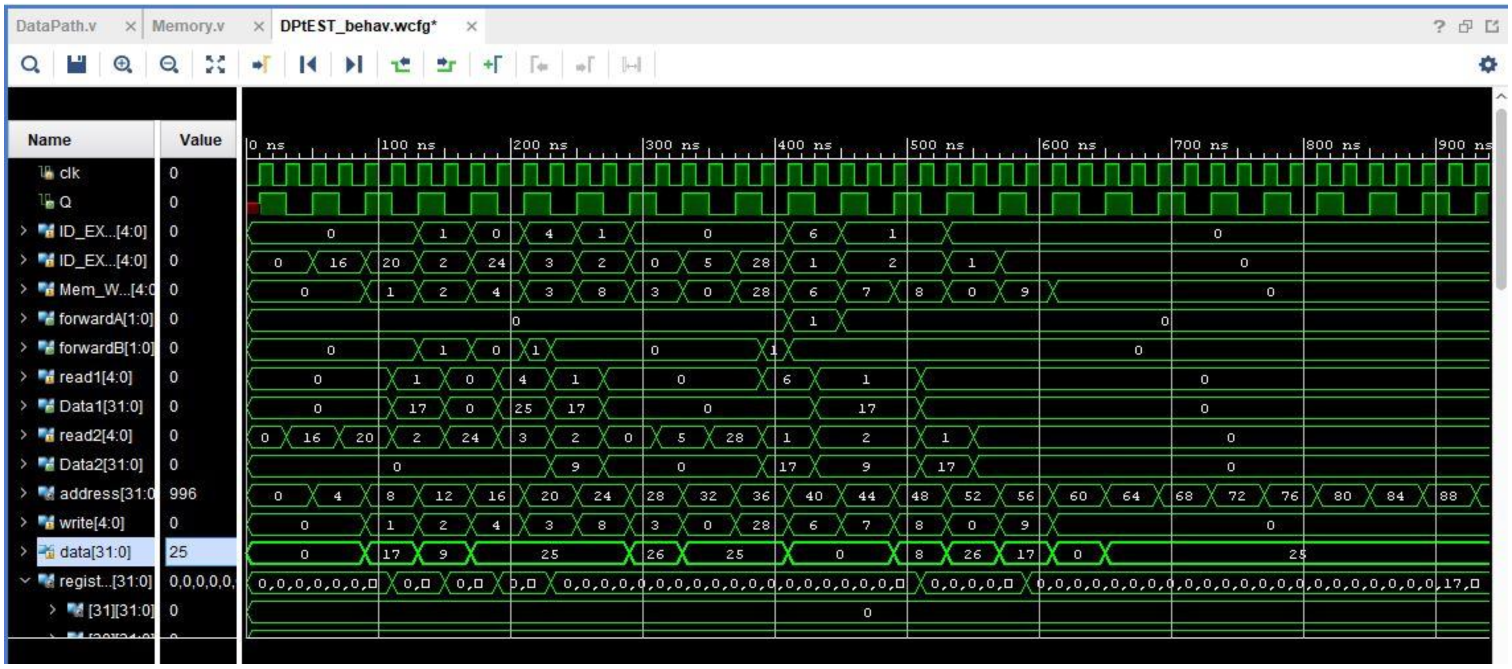
Results:

We tested the data-path in each stage while implementing, and you can find the tests folder attached in the zipped file. However, we constructed a final test to make sure that our pipelined can handle both types of Hazzard. Here is our test in assembly code bellow, followed by a screenshot of our simulation.

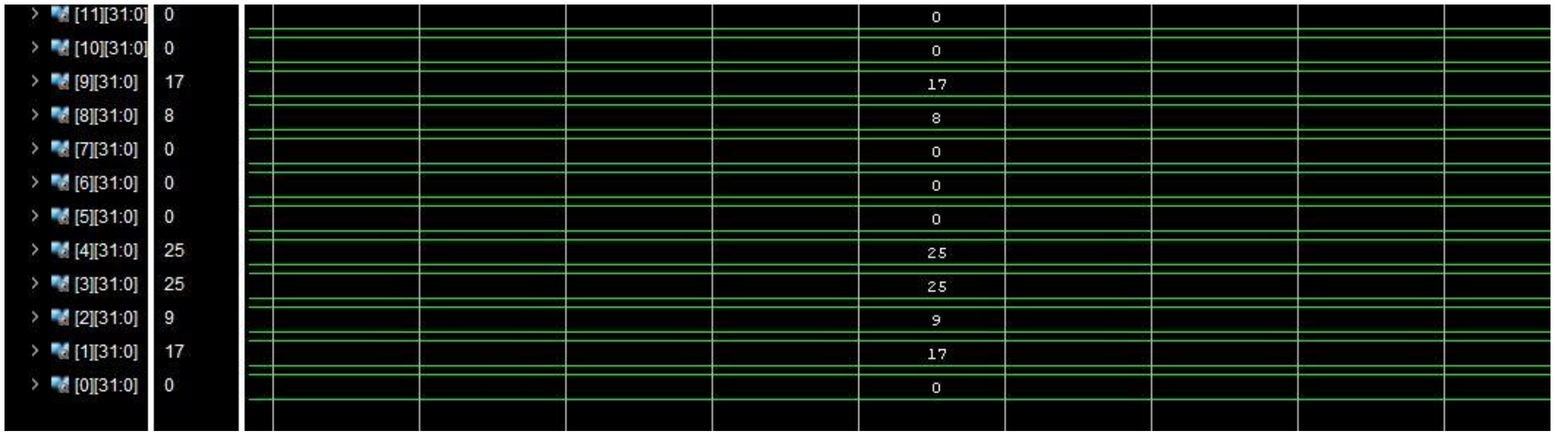
assembly

mem[400] = 17 , mem[404]=9, mem[408]=25

```
lw x1, 400(x0)
lw x2, 404(x0)
or x4, x1,x2
lw x3, 408(x0)
beq x4, x3, lable
add x3, x1, x2
lable:
add x5, x3, x2
sw x5, 412(x0)
lw x6, 412(x0)
and x7, x6, x1
sub x8, x1, x2
add x0, x1, x2
add x9, x0, x1
add x0, x0, x0
```



noticing the data stored in the register file we can confirm that the data-path solved both data and control hazard.



```
[11][31:0]  0
[10][31:0]  0
[9][31:0]   17
[8][31:0]   8
[7][31:0]   0
[6][31:0]   0
[5][31:0]   0
[4][31:0]   25
[3][31:0]   25
[2][31:0]   9
[1][31:0]   17
[0][31:0]   0
```

The results were right, and it flushes the instruction in the pipelined perfectly.

Note:

You can add from test folder test that you want to run to the memory.

Here are the Tests in assembly:

#Data_Dependency_test:

lw x1, 400(x0)

add x4, x1, x0

lw x2, 404(x0)

add x5, x4, x2

sub x6, x5, x2

and x6, x2, x3

#load_Dependency_test:

```
lw x1, 400(x0)
lw x2, 404(x0)
lw x3, 408(x0)
or x4, x1,x2
add x5, x1,x3
and x6, x2,x3
add x7, x1,x2
or x8, x2,x3
sw x4, 412(x0)
and x9, x4, x1
sub x10, x4, x1
add x11, x1,x5
lw x12, 412(x0)
xor x13, x4,x5
```

#overall_Dependency_test:

```
lw x1, 400(x0)
lw x2, 404(x0)
or x4, x1,x2
lw x3, 408(x0)
beq x4, x3, lable
add x3, x1, x2
lable:
add x5, x3, x2
sw x5, 412(x0)
lw x6, 412(x0)
and x7, x6, x1
```

```
sub x8, x1, x2
add x0, x1, x2
add x9, x0, x1
add x0, x0, x0
```