Team Nova

16.05.2020

Mohamed Ahmed Helmy El-Sayed	43-9565
Ramez Mohamed Ibrahim Elmasry	43-10324
Omar Abdelhamid Ismaeil Youssef Altobgy	43-12268
Fakhreldin Hussein Soliman Taraby	43-11215
George Nicola	43-1355

Overview

We implemented our architecture with specific features that help in drawing out the best performance possible according to our project proposal choices. Our architecture is implemented with reference to Harvard architecture where we can access both data and instruction memories simultaneously. The size of both data and instruction memories is 1024*16-bit. The total number of registers used is 16 registers, where all of them are general-purpose registers except that there is the zero register ($0) is used as an offset in Load word and Store word instructions. We implemented the instruction set 2. In order to implement instruction set 2, we had to create a new instruction format that suits our architecture implementation approach. Each Instruction size is 16 bits, and according to the type of the instruction, it goes through the datapath differently. The following table shows the different types of instructions and how we divide the instructions according to its type:

Instruction Format

R-Type

I-Type
(And/Add immediate)

Branch
(Branch on equal/less than)

Load Word

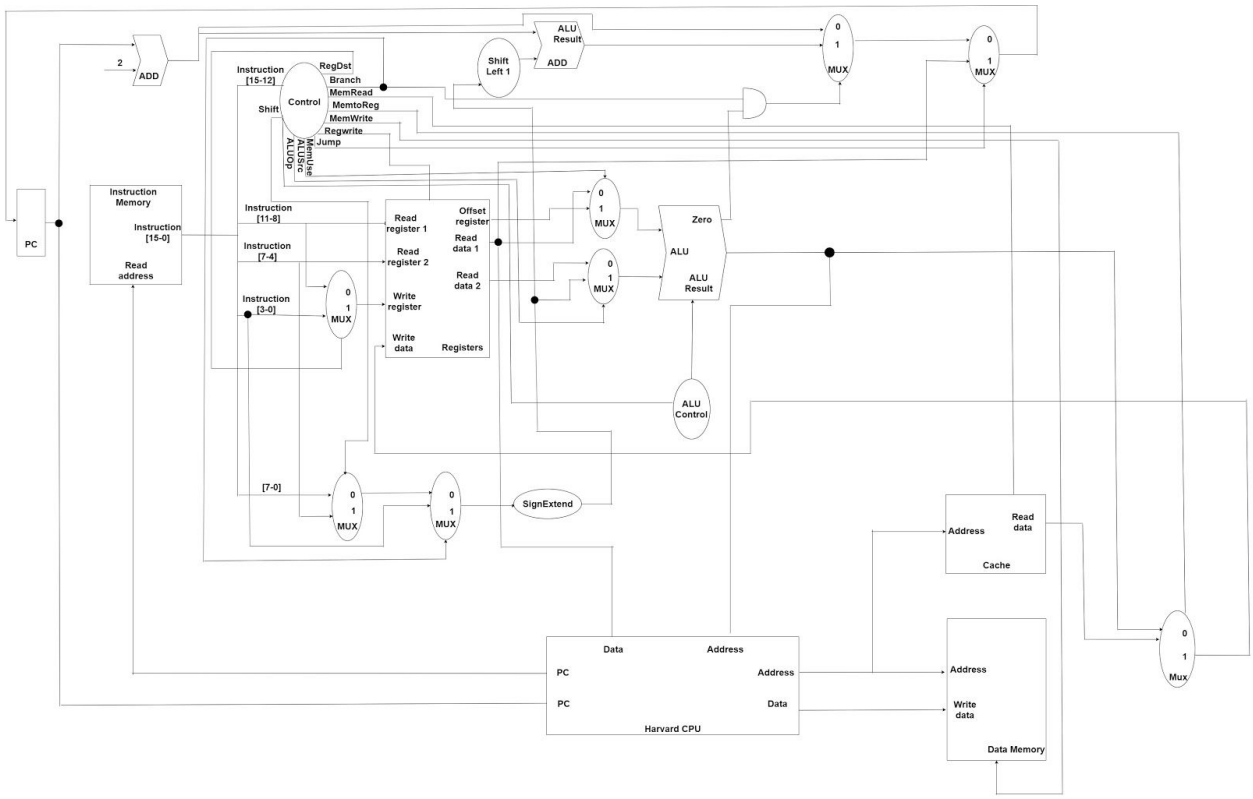
Store Word

Shift
(Shift Left/Right)

A side note is that the 8-bit immediate part in the I-type and branch instructions, also the 8-bit address part in load word and store word instructions are written as binary signed 2's complement.

We have created a cache in the data memory. The cache uses the direct-mapped organization, thus the replacement policy is built-in which means we did not have to use a specific replacement policy.

Then, for our datapath it goes as follows:

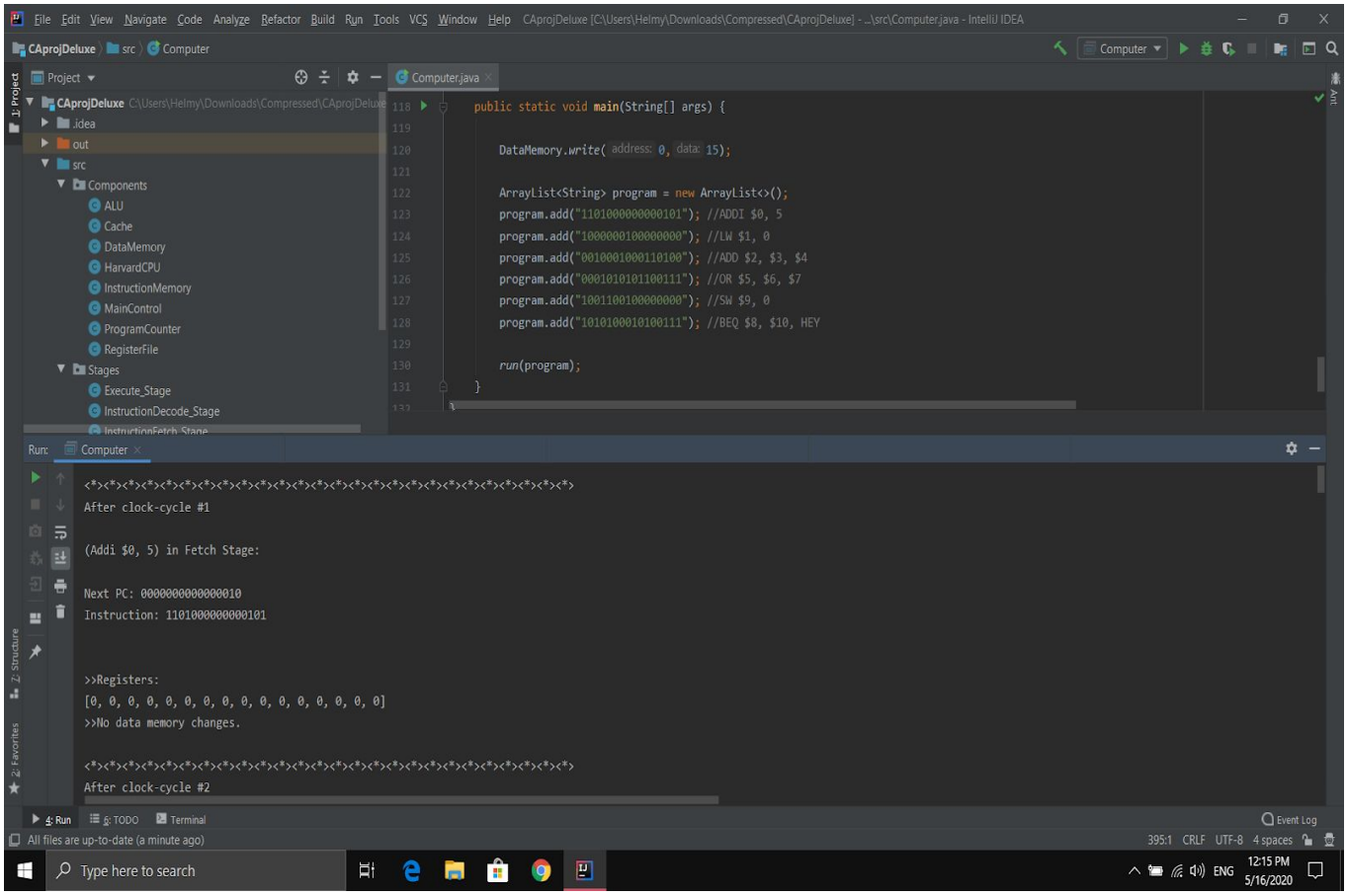


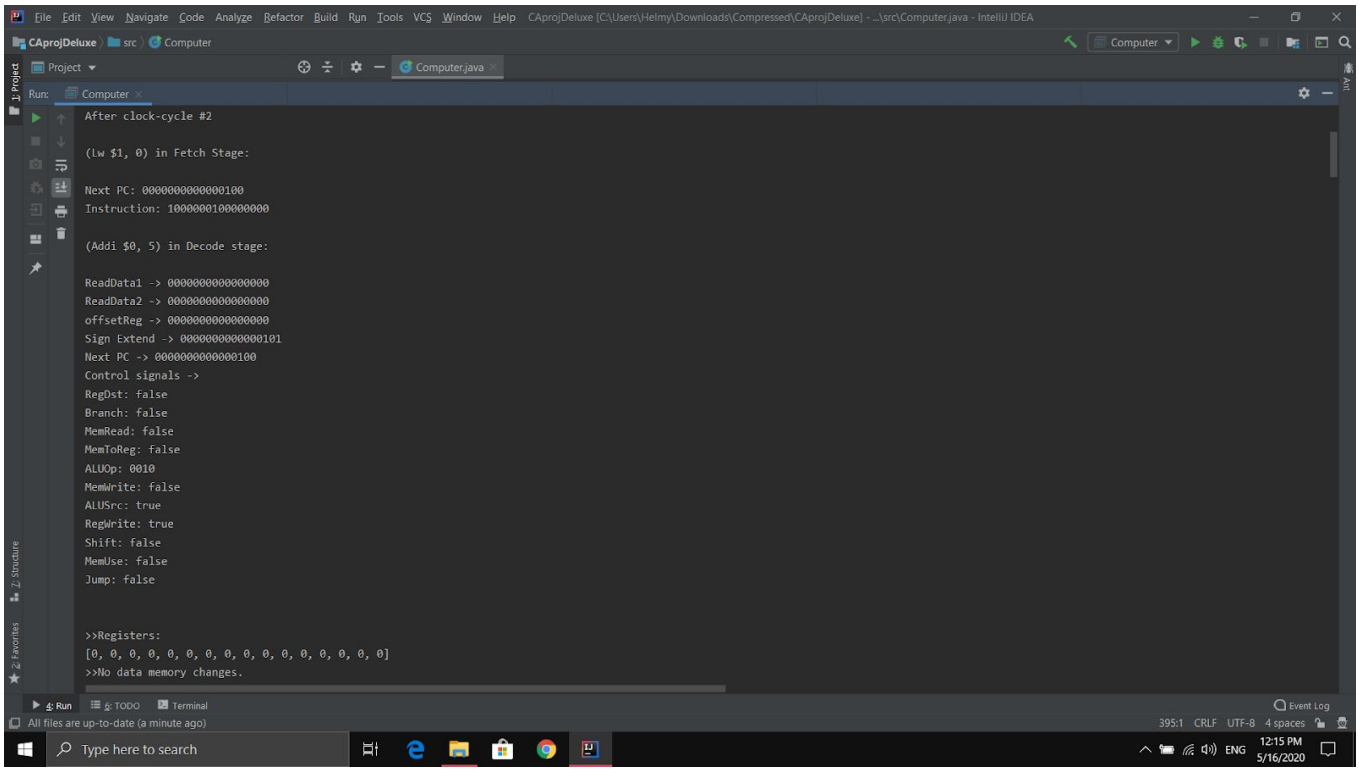
We can see from the graphical representation above that all the required components which are Instruction Memory, Data Memory + Cache, Register File, Program Counter, Arithmetic Logic Unit, Control Unit exist in the datapath. We added a new component however which is the Harvard CPU component which manages reading/writing data from/to data memory and its cache and setting the program counter value.

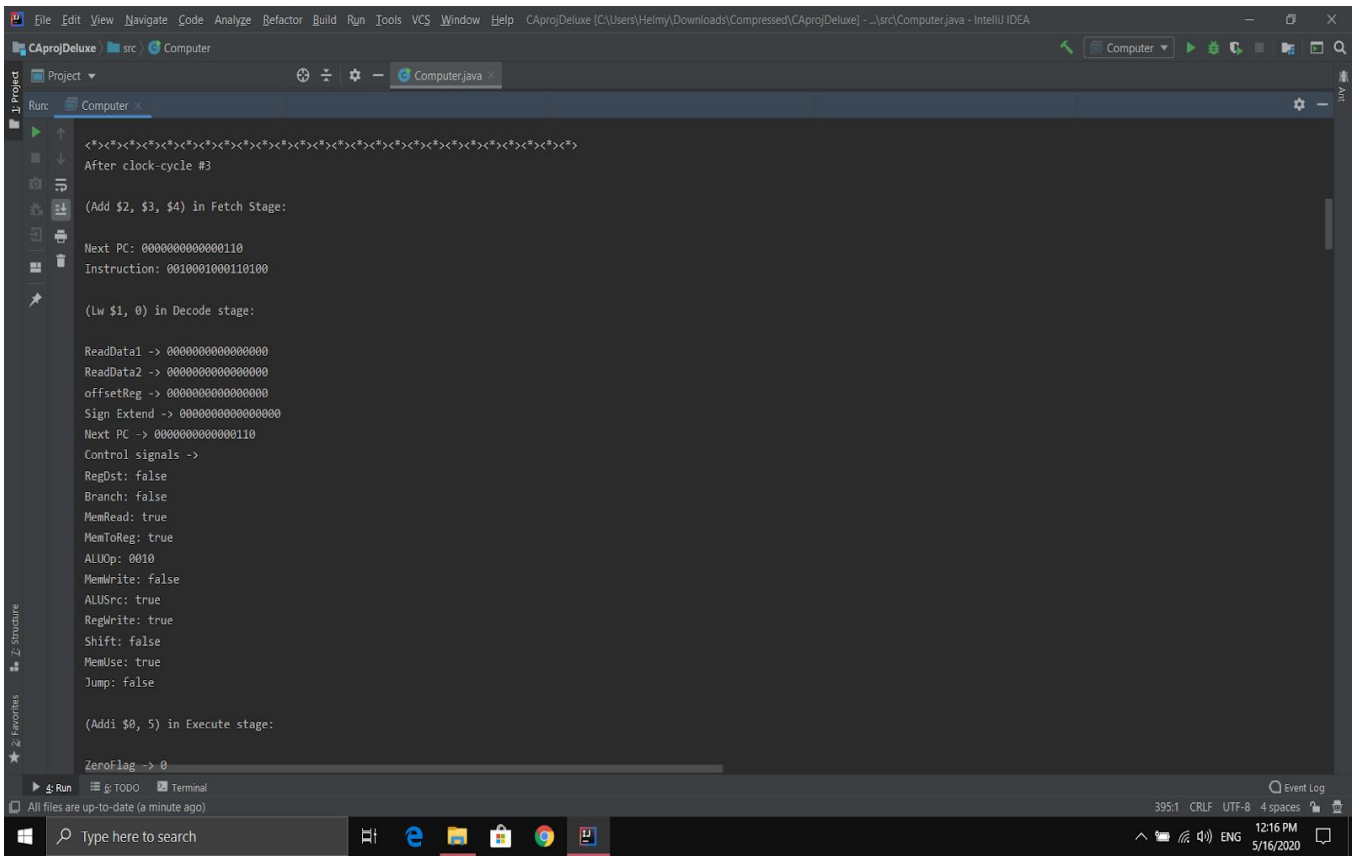
Finally, for our code structure, The code contains two packages (Components & Stages) and the main class (Computer). (Components) package has all the modules, And (Stages)

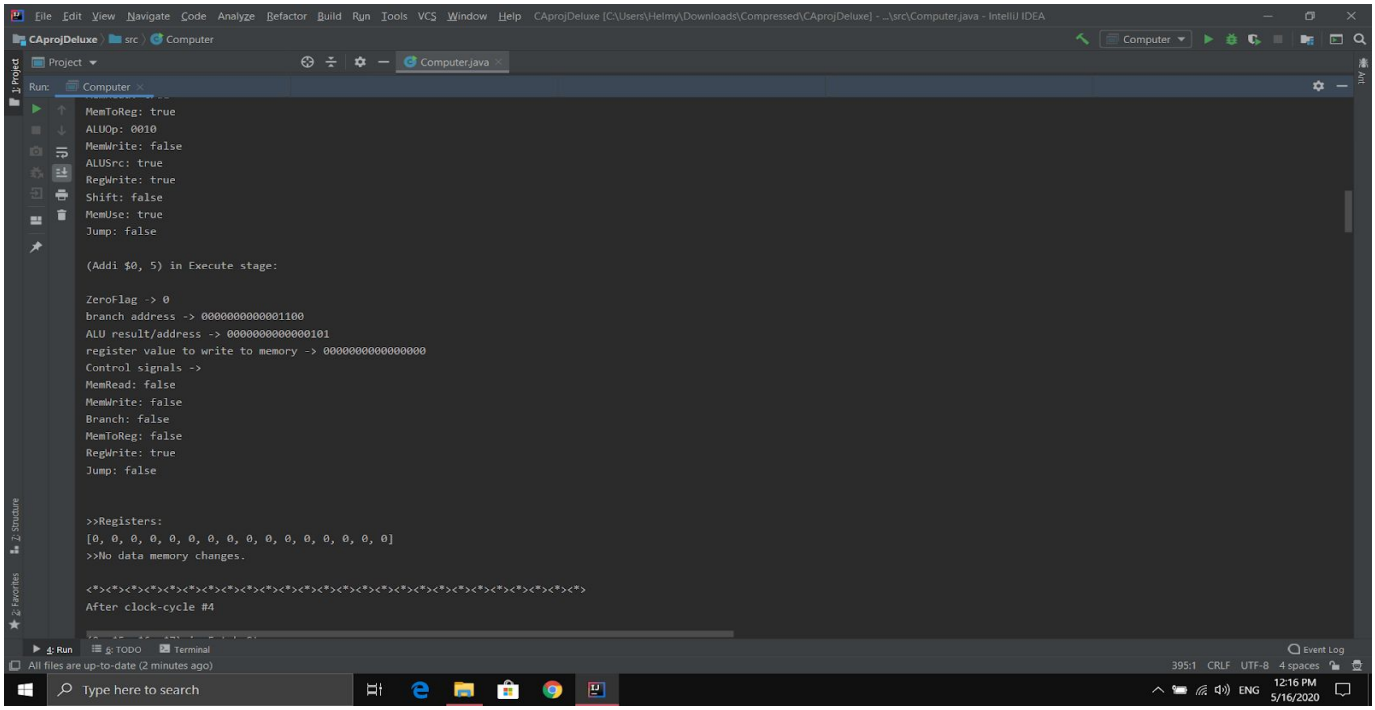
has all the stages (InstructionFetch - InstructionDecode - Execute - MemoryAccess - WriteBack). The program begins as an ArrayList of instructions which is passed to a run method in the main class (Computer). The run method loads the program to the instruction memory, then a run helper method is called which performs pipelining. Each instruction has to pass through all five stages. The program terminates after the last instruction to be executed goes through the (WriteBack) stage.

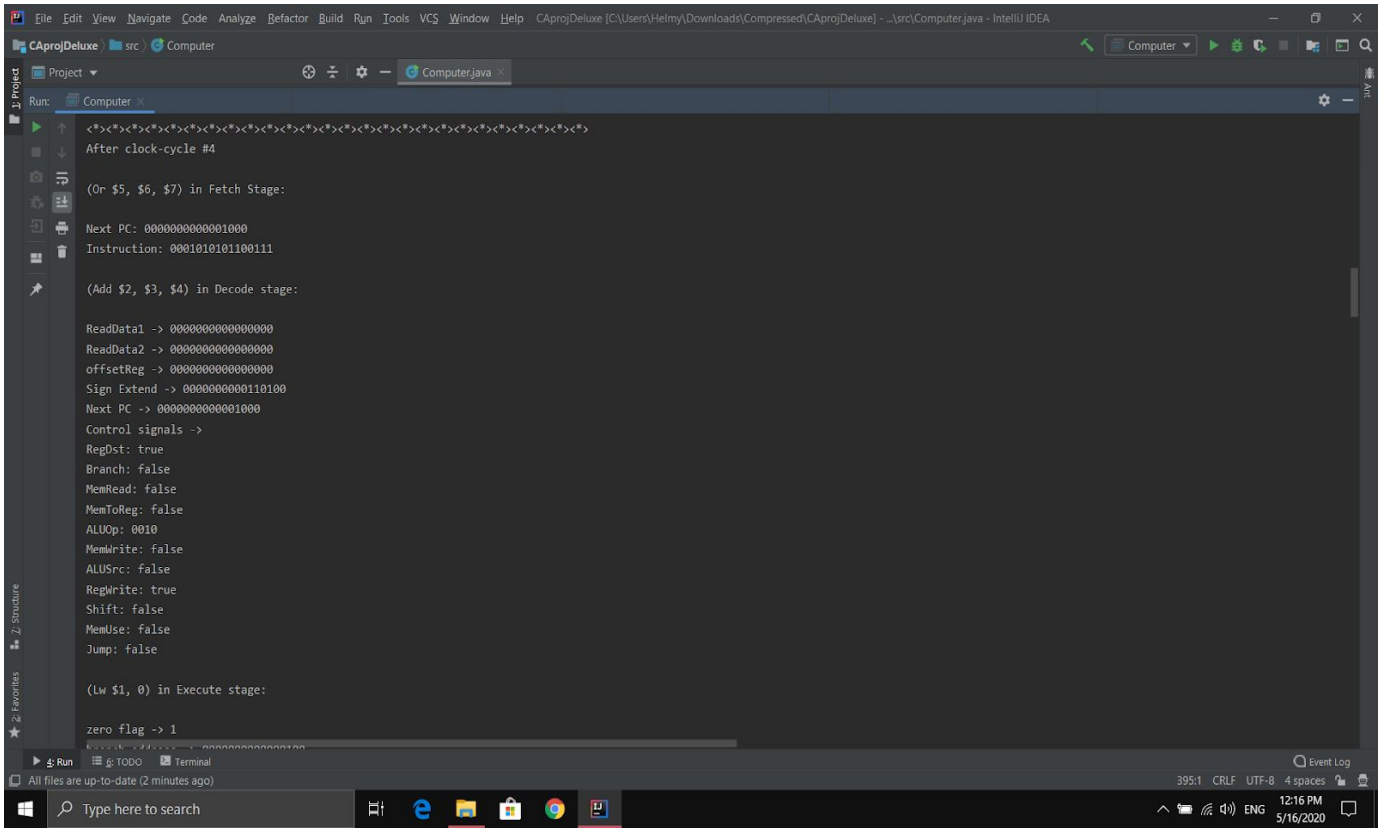
Our output is something like the following after loading a certain set of instructions

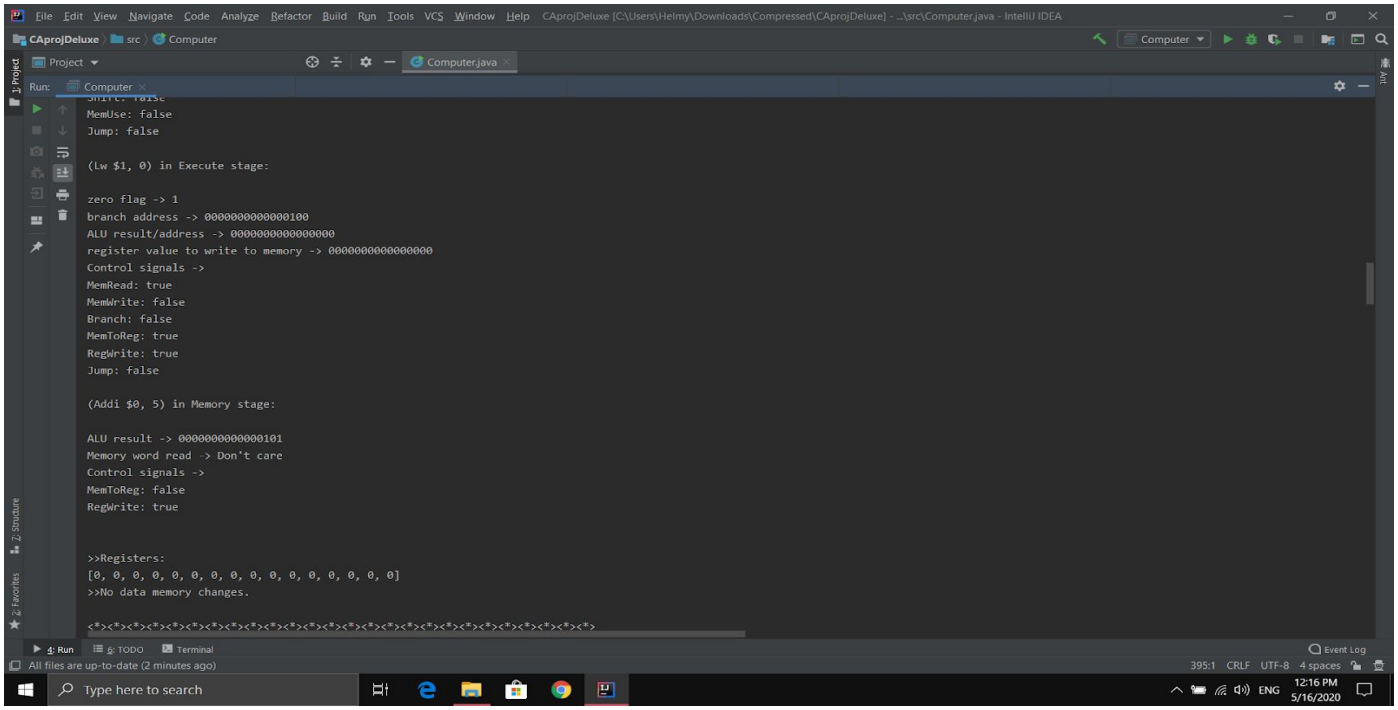


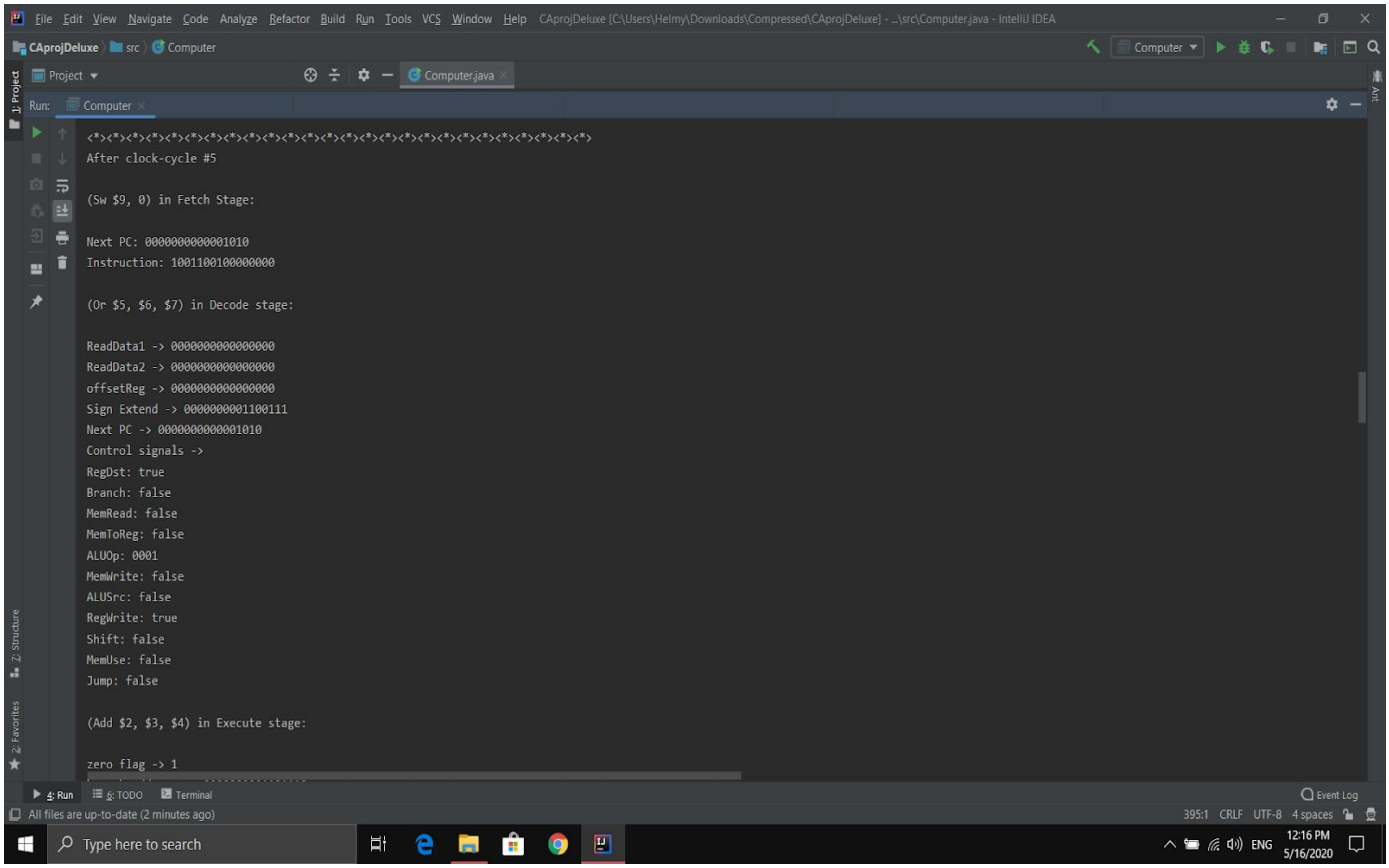

After clock-cycle #2
(Lw $1, 0) in Fetch Stage:
Next PC: 0000000000000100
Instruction: 1000000100000000
(Addi $0, 5) in Decode stage:
ReadData1 -> 0000000000000000
ReadData2 -> 0000000000000000
offsetReg -> 0000000000000000
Sign Extend -> 0000000000000101
Next PC -> 0000000000000100
Control signals ->
RegDst: false
Branch: false
MemRead: false
MemToReg: false
ALUOp: 0010
MemWrite: false
ALUSrc: true
RegWrite: true
Shift: false
MemUse: false
Jump: false
>>Registers:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.

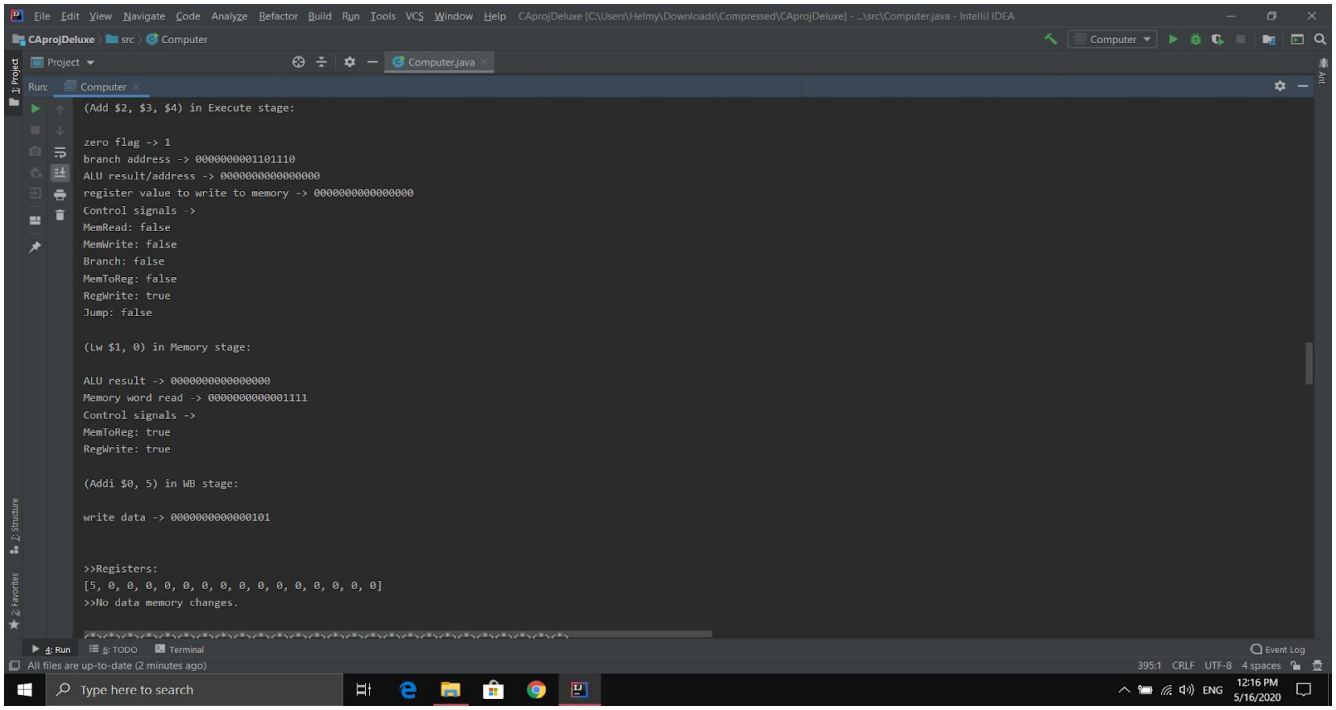

<*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*><*>
After clock-cycle #3
(Add $2, $3, $4) in Fetch Stage:
Next PC: 0000000000000110
Instruction: 0010001000110100
(Lw $1, 0) in Decode stage:
ReadData1 -> 0000000000000000
ReadData2 -> 0000000000000000
offsetReg -> 0000000000000000
Sign Extend -> 0000000000000000
Next PC -> 0000000000000110
Control signals ->
RegDst: false
Branch: false
MemRead: true
MemToReg: true
ALUOp: 0010
MemWrite: false
ALUSrc: true
RegWrite: true
Shift: false
MemUse: true
Jump: false
(Addi $0, 5) in Execute stage:
ZeroFlag -> 0

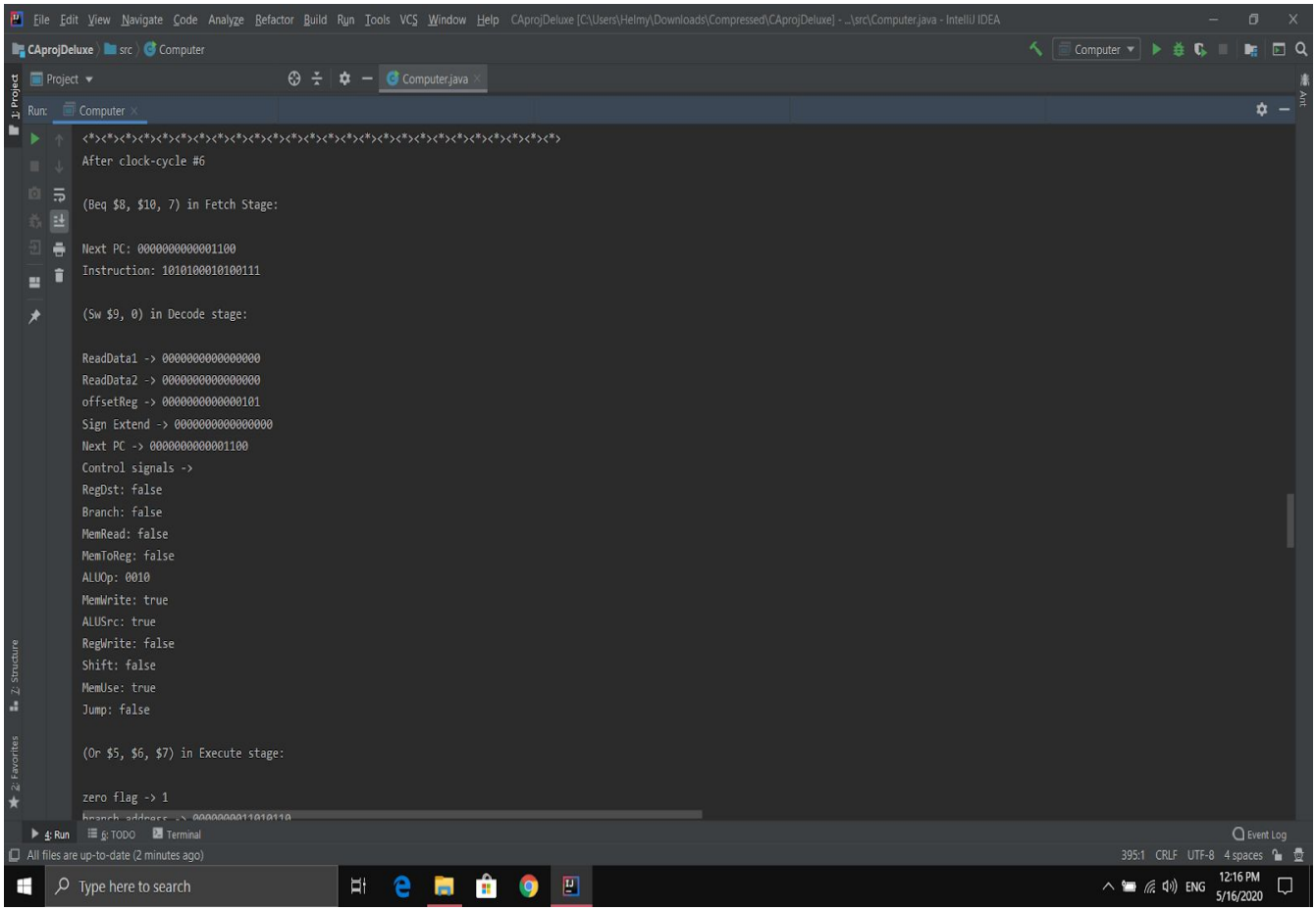

MemToReg: true
ALUOp: 0010
MemWrite: false
ALUSrc: true
RegWrite: true
Shift: false
MemUse: true
Jump: false
(Addi $0, 5) in Execute stage:
ZeroFlag -> 0
branch address -> 0000000000001100
ALU result/address -> 0000000000000101
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: false
Branch: false
MemToReg: false
RegWrite: true
Jump: false
>>Registers:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.
After clock-cycle #4

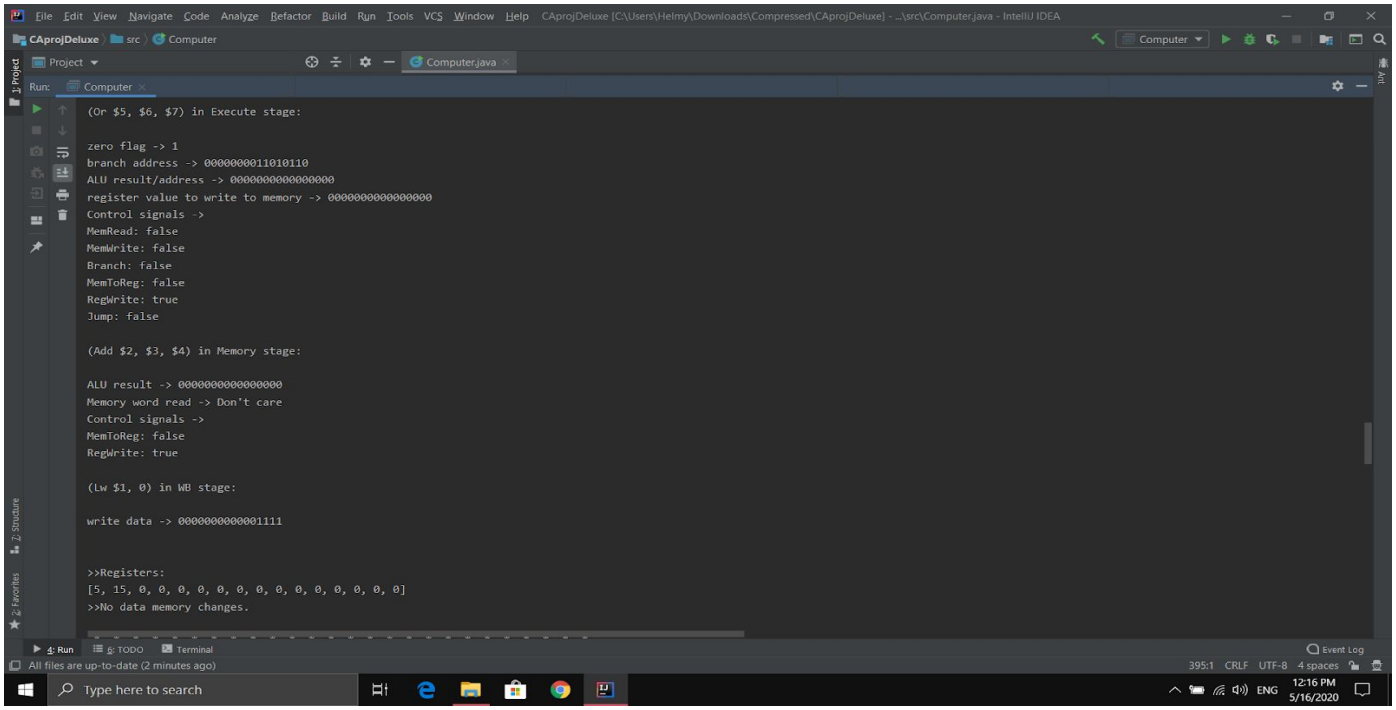

After clock-cycle #4
(Or $5, $6, $7) in Fetch Stage:
Next PC: 0000000000001000
Instruction: 00010101011001111
(Add $2, $3, $4) in Decode stage:
ReadData1 -> 0000000000000000
ReadData2 -> 0000000000000000
offsetReg -> 0000000000000000
Sign Extend -> 0000000000110100
Next PC -> 0000000000001000
Control signals ->
RegDst: true
Branch: false
MemRead: false
MemToReg: false
ALUOp: 0010
MemWrite: false
ALUSrc: false
RegWrite: true
Shift: false
MemUse: false
Jump: false
(Lw $1, 0) in Execute stage:
zero flag -> 1

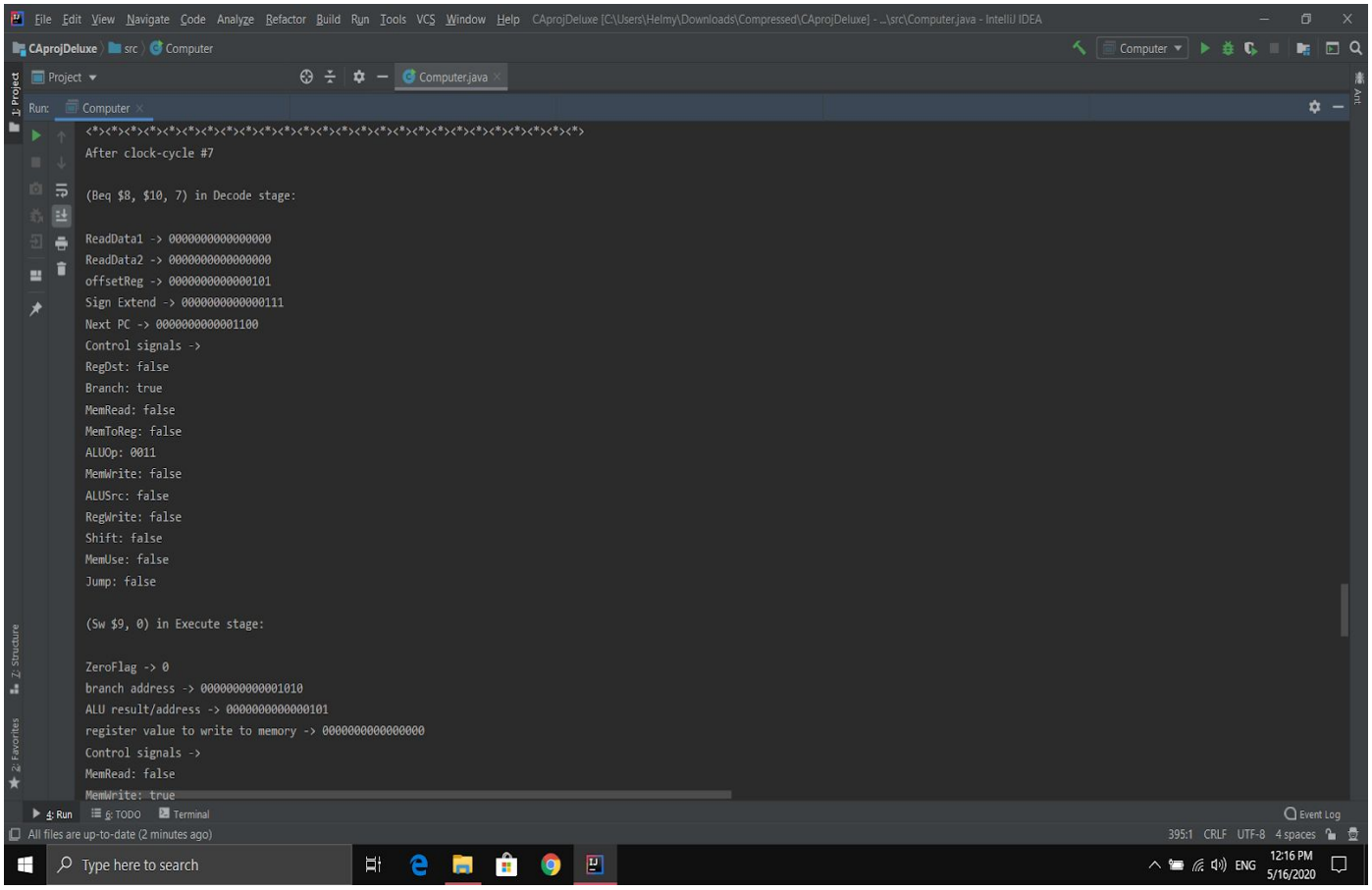

MemUse: false
Jump: false
(Lw $1, 0) in Execute stage:
zero flag -> 1
branch address -> 0000000000000100
ALU result/address -> 0000000000000000
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: true
MemWrite: false
Branch: false
MemToReg: true
RegWrite: true
Jump: false
(Addi $0, 5) in Memory stage:
ALU result -> 0000000000000101
Memory word read -> Don't care
Control signals ->
MemToReg: false
RegWrite: true
>>Registers:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.

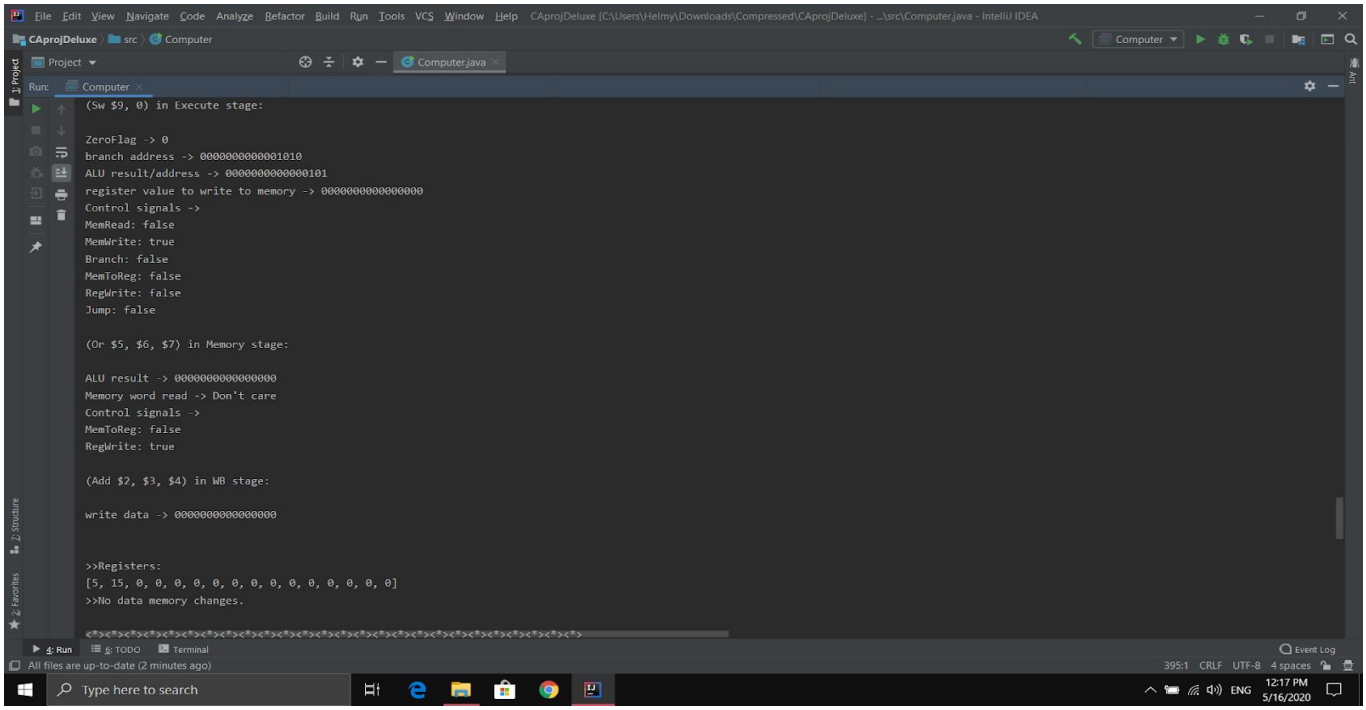

After clock-cycle #5
(Sw $9, 0) in Fetch Stage:
Next PC: 0000000000001010
Instruction: 1001100100000000
(Or $5, $6, $7) in Decode stage:
ReadData1 -> 0000000000000000
ReadData2 -> 0000000000000000
offsetReg -> 0000000000000000
Sign Extend -> 0000000001100111
Next PC -> 0000000000001010
Control signals ->
RegDst: true
Branch: false
MemRead: false
MemToReg: false
ALUOp: 0001
MemWrite: false
ALUSrc: false
RegWrite: true
Shift: false
MemUse: false
Jump: false
(Add $2, $3, $4) in Execute stage:
zero flag -> 1

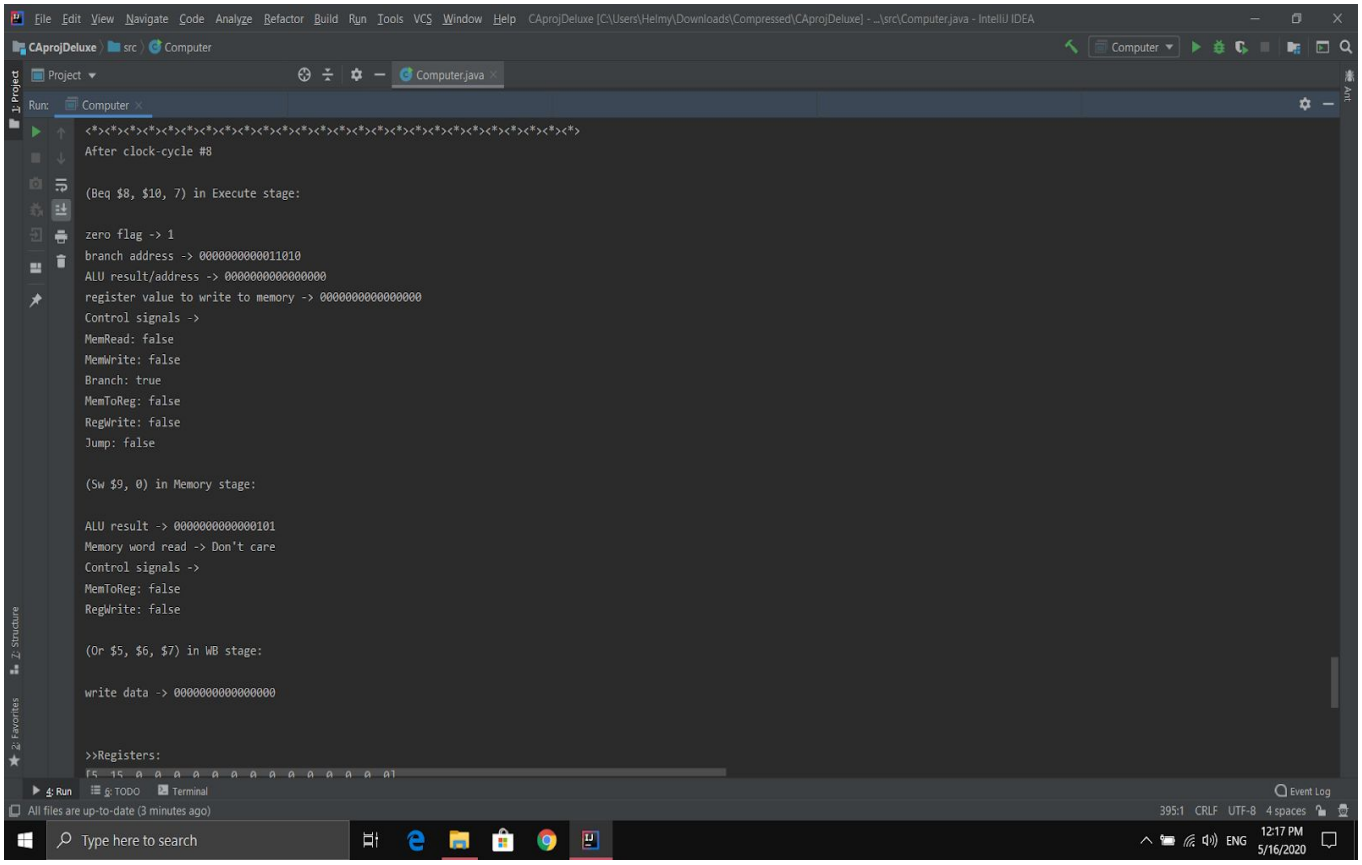

(Add $2, $3, $4) in Execute stage:
zero flag -> 1
branch address -> 0000000001101110
ALU result/address -> 0000000000000000
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: false
Branch: false
MemToReg: false
RegWrite: true
Jump: false
(Lw $1, 0) in Memory stage:
ALU result -> 0000000000000000
Memory word read -> 0000000000001111
Control signals ->
MemToReg: true
RegWrite: true
(Addi $0, 5) in WB stage:
write data -> 0000000000000101
>>Registers:
[5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.

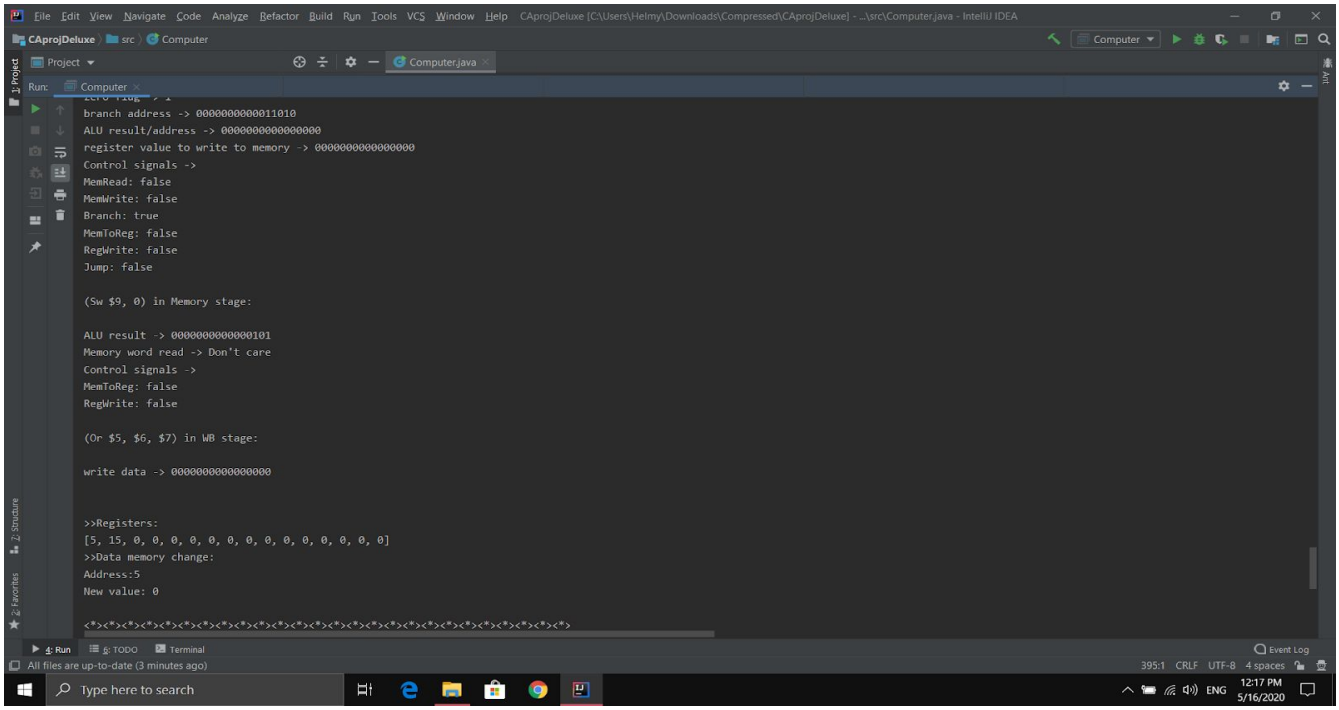

After clock-cycle #6
(Beq $8, $10, 7) in Fetch Stage:
Next PC: 0000000000001100
Instruction: 10101000010100111
(Sw $9, 0) in Decode stage:
ReadData1 -> 0000000000000000
ReadData2 -> 0000000000000000
offsetReg -> 0000000000000101
Sign Extend -> 0000000000000000
Next PC -> 0000000000001100
Control signals ->
RegDst: false
Branch: false
MemRead: false
MemToReg: false
ALUOp: 0010
MemWrite: true
ALUSrc: true
RegWrite: false
Shift: false
MemUse: true
Jump: false
(Or $5, $6, $7) in Execute stage:
zero flag -> 1

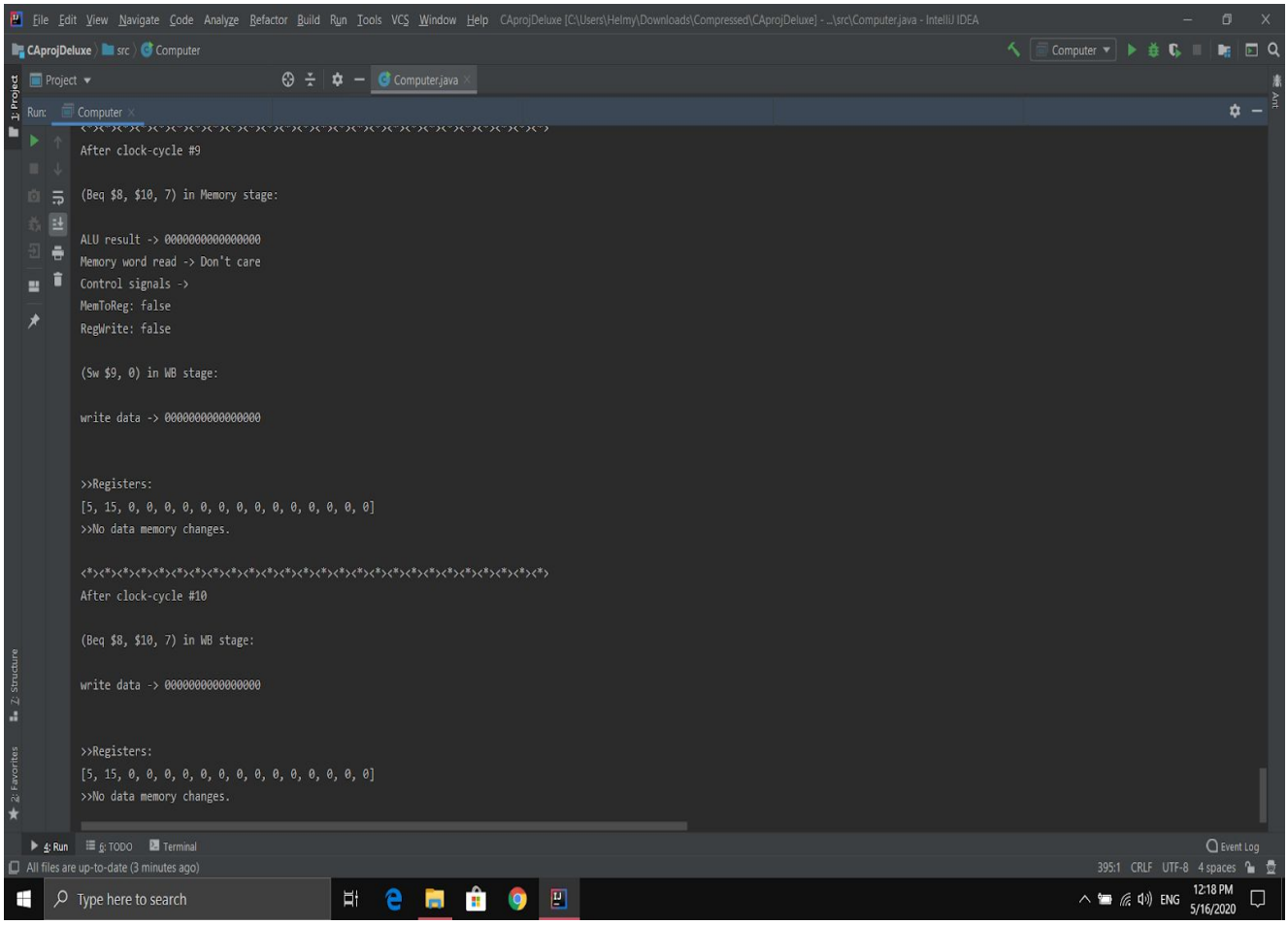

(Or $5, $6, $7) in Execute stage:
zero flag -> 1
branch address -> 0000000011010110
ALU result/address -> 0000000000000000
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: false
Branch: false
MemToReg: false
RegWrite: true
Jump: false
(Add $2, $3, $4) in Memory stage:
ALU result -> 0000000000000000
Memory word read -> Don't care
Control signals ->
MemToReg: false
RegWrite: true
(Lw $1, 0) in WB stage:
write data -> 0000000000001111
>>Registers:
[5, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.


After clock-cycle #7
(Beq $8, $10, 7) in Decode stage:
ReadData1 -> 0000000000000000
ReadData2 -> 0000000000000000
offsetReg -> 0000000000000101
Sign Extend -> 0000000000000111
Next PC -> 0000000000001100
Control signals ->
RegDst: false
Branch: true
MemRead: false
MemToReg: false
ALUOp: 0011
MemWrite: false
ALUSrc: false
RegWrite: false
Shift: false
MemUse: false
Jump: false
(Sw $9, 0) in Execute stage:
ZeroFlag -> 0
branch address -> 0000000000001010
ALU result/address -> 0000000000000101
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: true


(Sw $9, 0) in Execute stage:
ZeroFlag -> 0
branch address -> 0000000000001010
ALU result/address -> 0000000000000101
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: true
Branch: false
MemToReg: false
RegWrite: false
Jump: false
(Or $5, $6, $7) in Memory stage:
ALU result -> 0000000000000000
Memory word read -> Don't care
Control signals ->
MemToReg: false
RegWrite: true
(Add $2, $3, $4) in WB stage:
write data -> 0000000000000000
>>Registers:
[5, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.


After clock-cycle #8
(Beq $8, $10, 7) in Execute stage:
zero flag -> 1
branch address -> 0000000000011010
ALU result/address -> 0000000000000000
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: false
Branch: true
MemToReg: false
RegWrite: false
Jump: false
(Sw $9, 0) in Memory stage:
ALU result -> 0000000000000101
Memory word read -> Don't care
Control signals ->
MemToReg: false
RegWrite: false
(Or $5, $6, $7) in WB stage:
write data -> 0000000000000000
>>Registers:


branch address -> 0000000000011010
ALU result/address -> 0000000000000000
register value to write to memory -> 0000000000000000
Control signals ->
MemRead: false
MemWrite: false
Branch: true
MemToReg: false
RegWrite: false
Jump: false
(Sw $9, 0) in Memory stage:
ALU result -> 0000000000000101
Memory word read -> Don't care
Control signals ->
MemToReg: false
RegWrite: false
(Or $5, $6, $7) in WB stage:
write data -> 0000000000000000
>>Registers:
[5, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>Data memory change:
Address:5
New value: 0


After clock-cycle #9
(Beq $8, $10, 7) in Memory stage:
ALU result -> 0000000000000000
Memory word read -> Don't care
Control signals ->
MemToReg: false
RegWrite: false
(Sw $9, 0) in WB stage:
write data -> 0000000000000000
>>Registers:
[5, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.
After clock-cycle #10
(Beq $8, $10, 7) in WB stage:
write data -> 0000000000000000
>>Registers:
[5, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>No data memory changes.