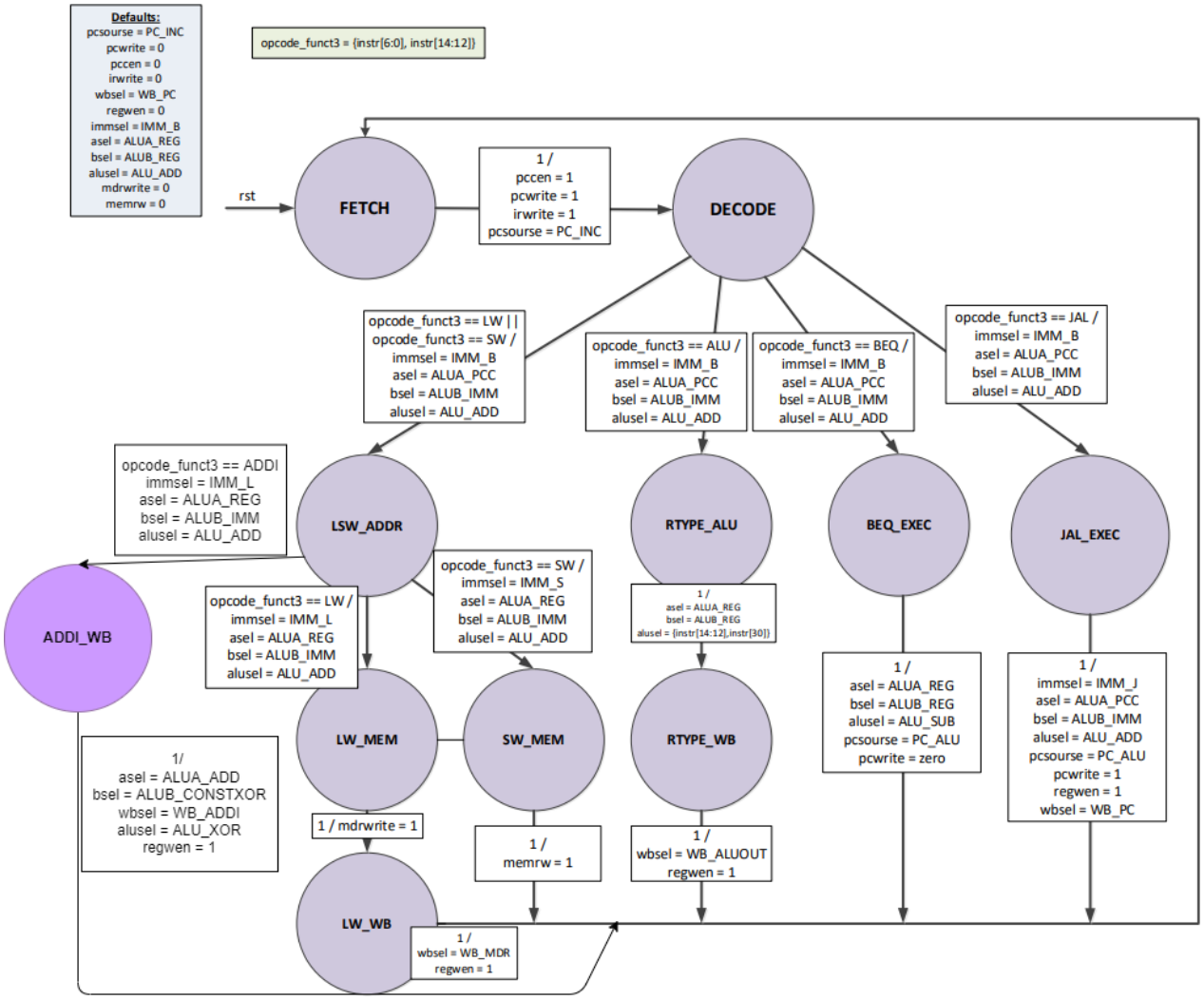
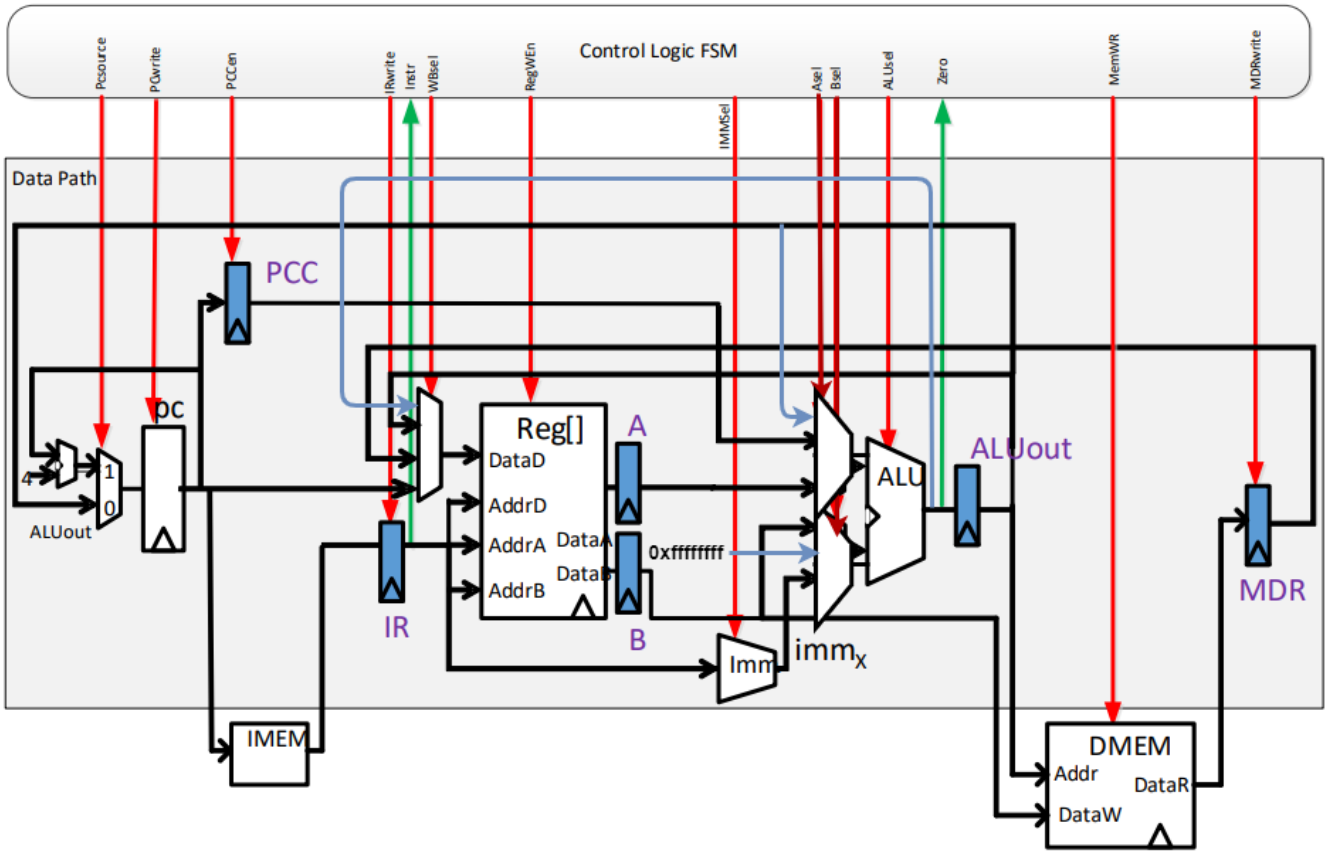
Simulation 3
Dry Part

שרה גריפית	341312304
טל שמיר	207223066

2.1: Updated diagrams



Simulation 3
Dry Part



The wires added are in light blue in the diagram above.

The following are the changes made to the processor to allow implementation of the action addi:

- The muxes selecting the ALU input were changed from 2->1 to 4->1.
- A wire was added that connects the ALU output to the 4->1 mux that determines the DataD input in the register.
- A wire was added from the output of ALUout to the 4->1 mux that determines input A of the ALU.
- A wire was added containing a constant value of 0xffffffff to the 4->1 mux that determines input B of the ALU.

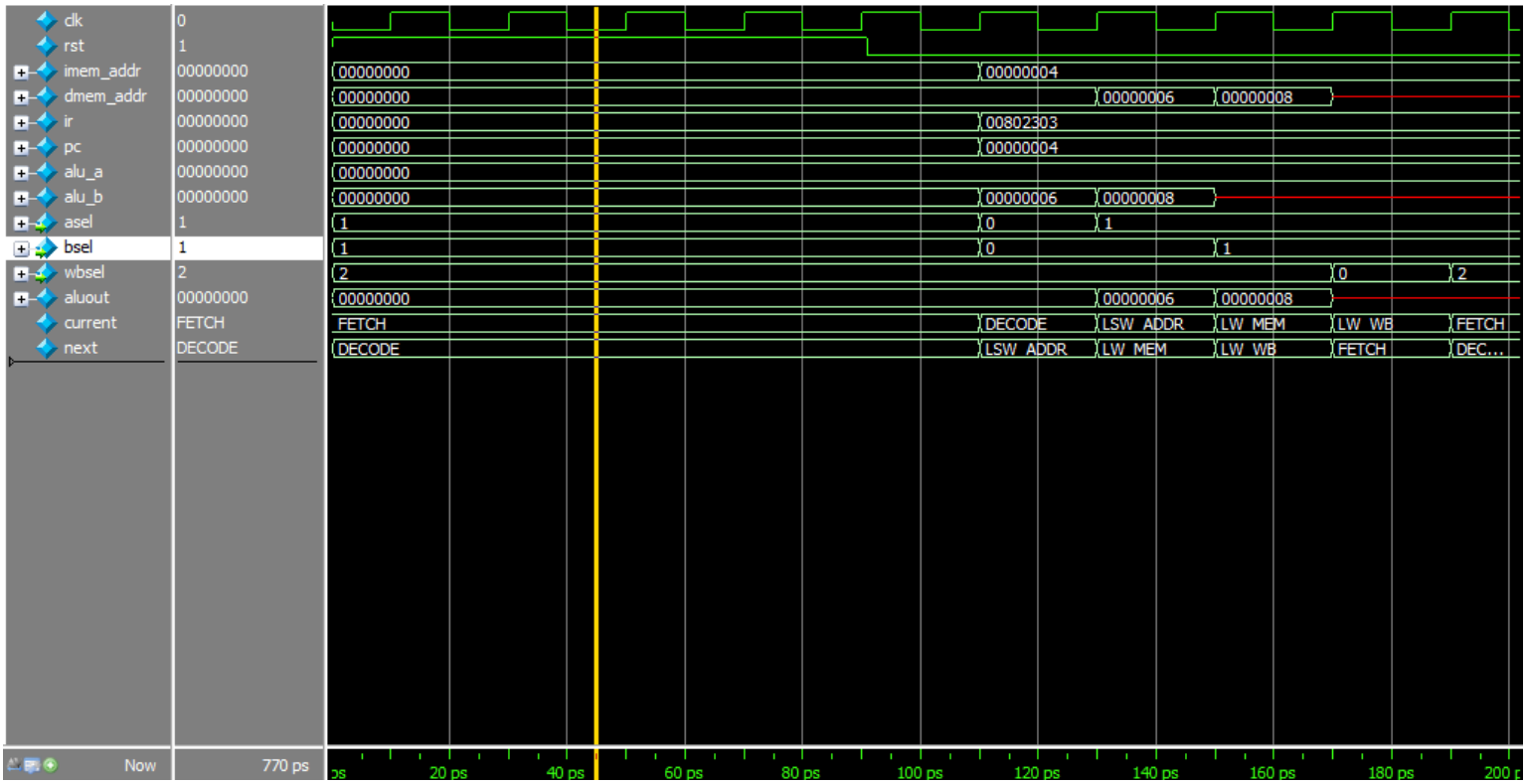
When addi is called, the processor does the following:

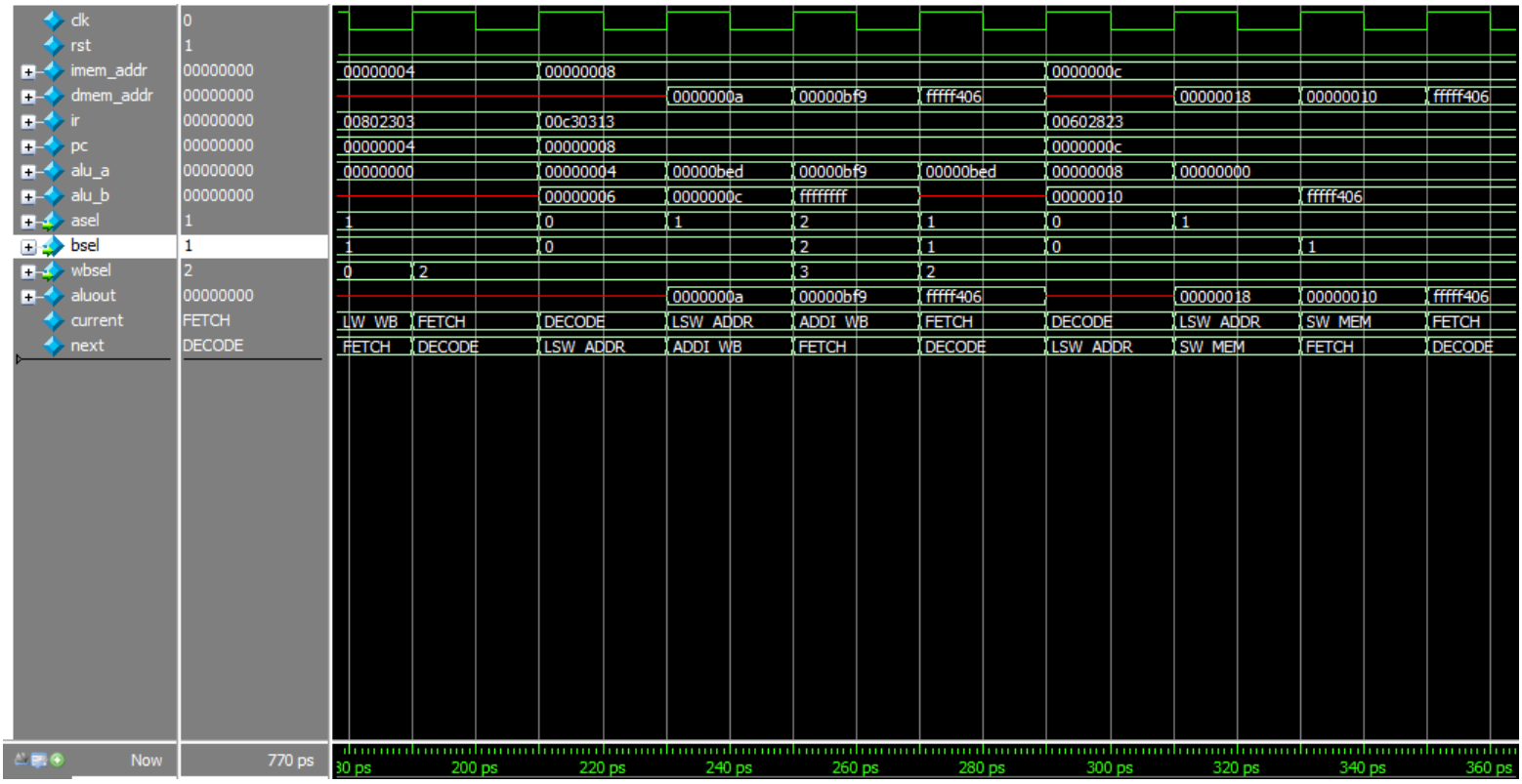
1. FETCH: The processor fetches the command from the pc (in our case addi).
2. DECODE: The processor decodes the command addi and performs the addition required for a jump/branch command. This is the case in any command the processor receives.
3. LSW_ADDR: When a lw command is received, this step adds an immediate given with the input A from the proper register. This is the required action for addi, and therefore this state was used. Thus, register A is added with the immediate given and the result is saved in ALUout.
4. ADDI_WB: the result saved in ALUout is sent as an input to the ALU through the mux that determines input A. The mux that determines the input B of the ALU receives the constant value from the wire added. Both values above are selected using the asel and bsel accordingly. The ALU receives the signal to perform XOR, and the output is sent to DataD in the registers through the light blue wire added after the ALU.

Simulation 3
Dry Part

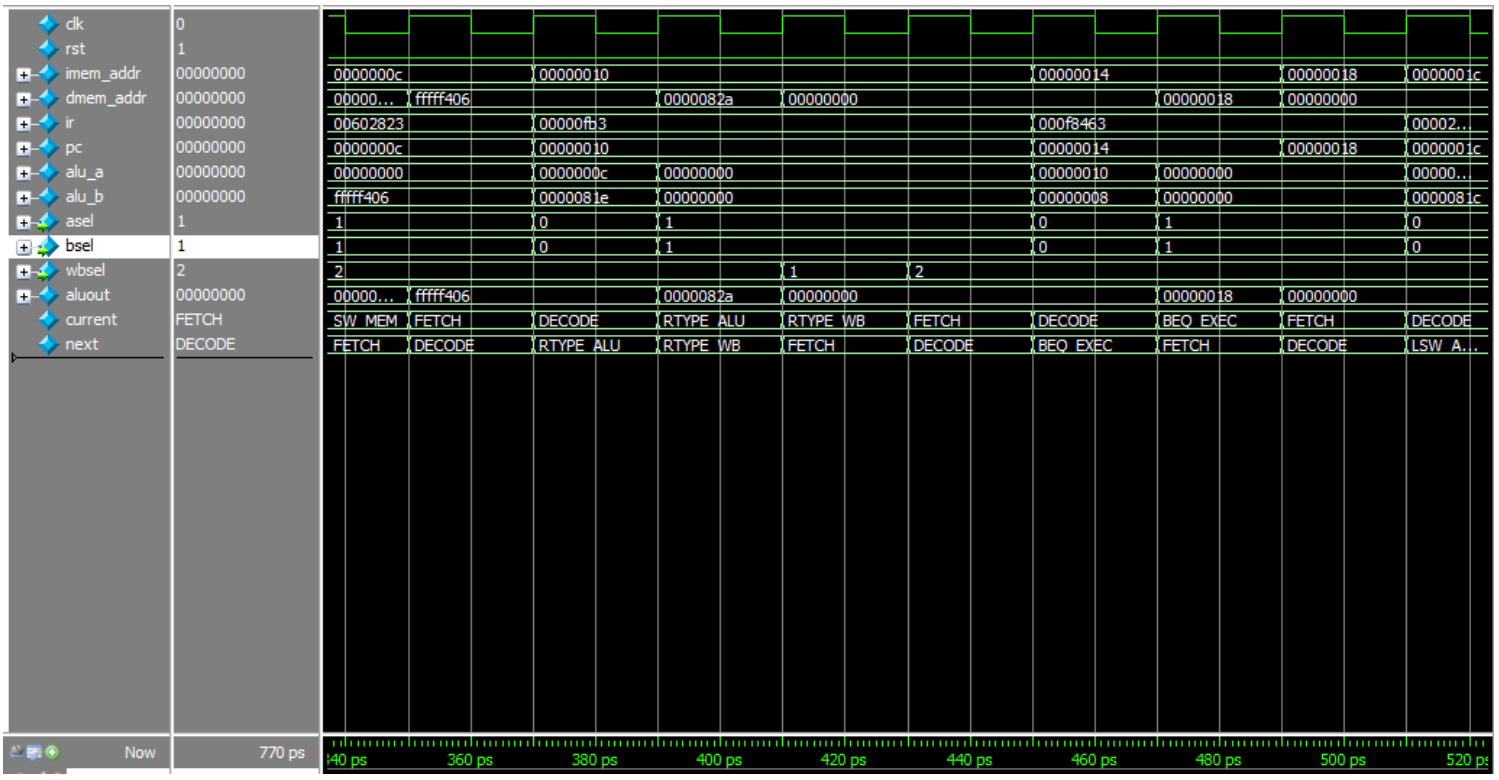
At the end of this process, the processor returns to the state FETCH to allow the next command to run.

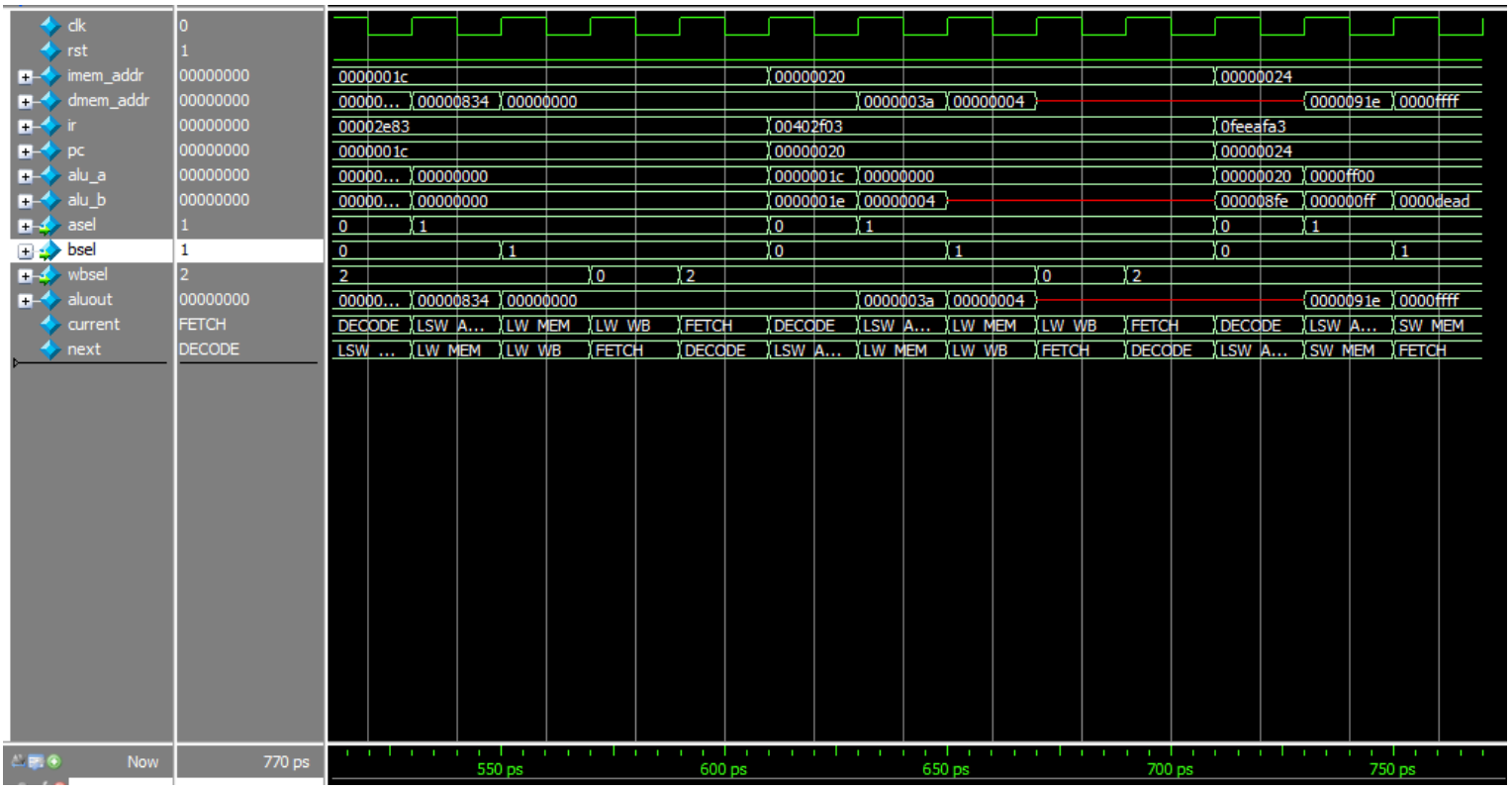
2.3: Results:





Simulation 3
Dry Part





The simulation starts with 5 clock cycles in which the reset has the value of 1. During this time, the current state is FETCH, and all outputs are 0 as required. After one clock cycle, the current state becomes DECODE and next state becomes LSW_ADDR, befitting the load word command fetched. As seen in the first photo, asel is ALUA_PCC and bsel is ALUB_IMM, as required by the processor to perform a jump or branch command. This is unchanged from the original processor given. After one

Simulation 3
Dry Part

clock cycle, the current state becomes LSW_ADDR. During this cycle, asel is ALUA_REG and bsel is ALUB_IMM. This is unchanged from the state diagram of the original processor. After one clock cycle, the current state becomes LW_MEM, during which a word is loaded from the appropriate DMEM address. After one clock cycle, the current state becomes LW_WB. As seen in the state diagram given, the word is written into the correct register. After this, the state becomes FETCH, and the processor fetches a new command.

The current command is now addi. After one clock cycle, the current state moves to DECODE. As seen in the second photo, asel is ALUA_PCC and bsel is ALUB_IMM, as required by the processor to perform a jump or branch command. This is unchanged from the original processor given. After one clock cycle, the current state becomes LSW_ADDR. During this cycle, asel is ALUA_REG and bsel is ALUB_IMM. The command addi must add the contents of register A with the given immediate. This is the action performed during state LSW_ADDR, and so therefore the processor moves to that state. From here, the processor moves to a new state, ADDI_WB, created specifically for implementing the command addi (as required in the assignment). During this state, asel is ALUA_ADD, bsel is ALUB_CONSTXOR, and wbsel is WB_ADDI. alusel is equal to ALU_XOR, allowing the processor to perform the required XOR on the result of adding register A with the immediate (which was performed during the previous state). regwen is equal to 1, allowing the processor to write the result to the proper register. From here, the current state becomes FETCH, and the processor fetches a new command.

The current command is now sw. After one clock cycle, the current state becomes DECODE. As seen in the second photo, asel is ALUA_PCC and bsel is ALUB_IMM, as required by the processor to perform a jump or branch command. This is unchanged from the original processor given. After one clock cycle, the current state becomes LSW_ADDR. During this cycle, asel is ALUA_REG and bsel is ALUB_IMM. This is unchanged from the state diagram of the original processor. After one clock cycle, the current state becomes SW_MEM, during which the word is stored in the appropriate DMEM address. After this, the current state becomes FETCH, allowing the processor to fetch the next command.

Hereon, the commands received are add, beq, lw, and sw. All these commands work as in the original processor given. The run finishes after a call of sw, during which the value required to stop the code is written into the memory. This is seen in the fourth photo.