****

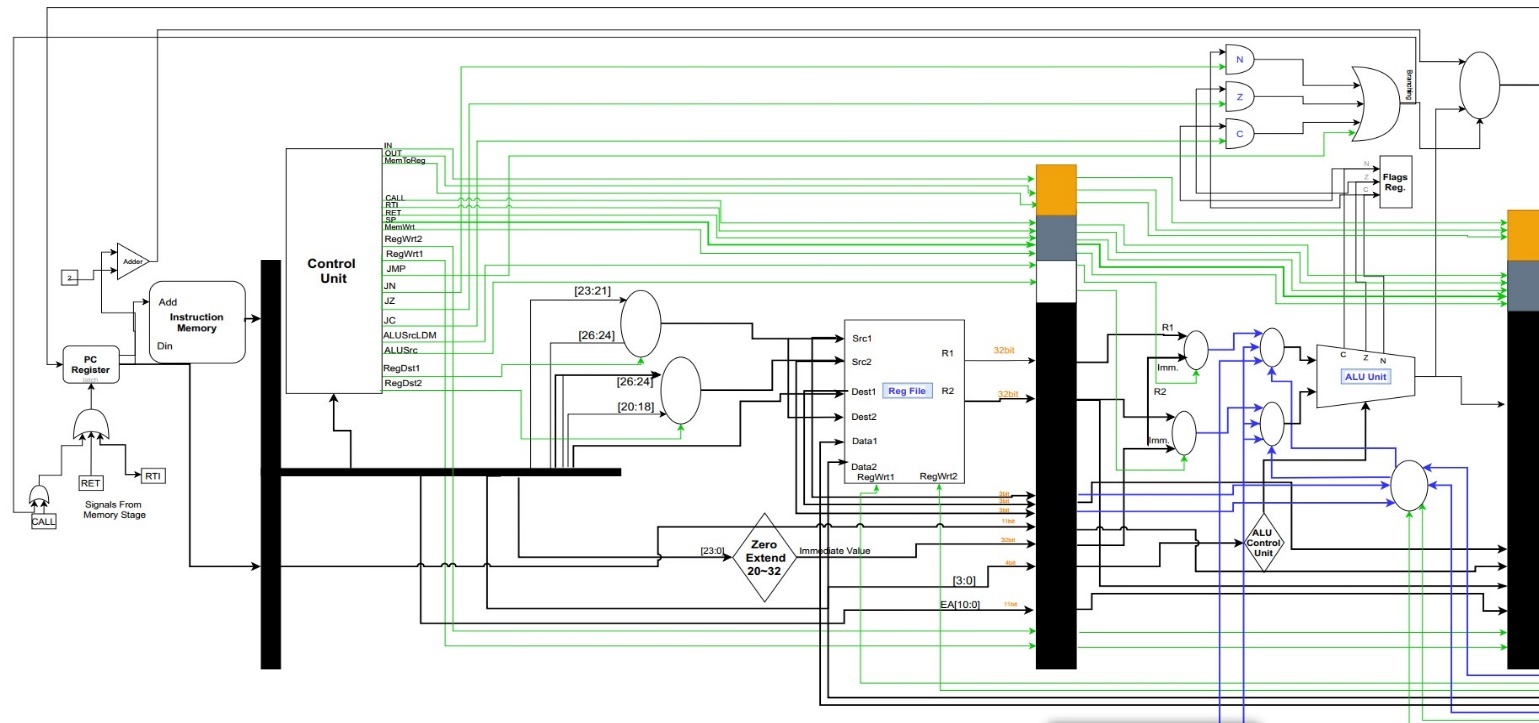
**CMPN301**

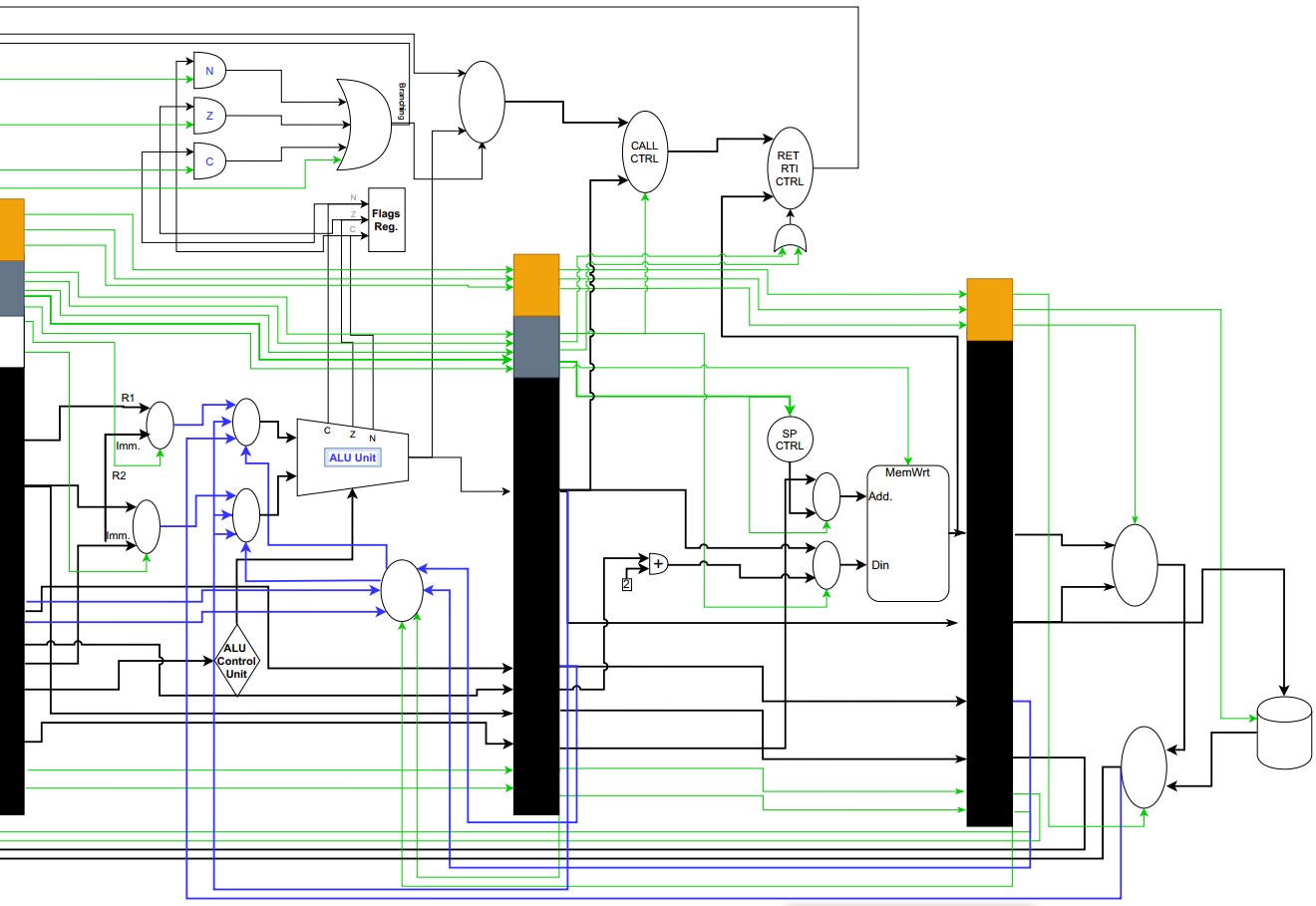
**Computer Architecture**

Project Report

**Mohamed Sayed Esmail**

**1162418**

**Processor Design**



**What is not Working…?**

1. Cache Memory:

**Primarily, the cache memory code was written and successfully compiled, but we didn’t consider the parallelism of the two caches and one memory. Therefore, the design failed to handle the hazard of the concurrent memory read and write and was replaced with the ordinary two separate memories with some modifications.**

1. RTI Instruction**:**

**The RTI instruction works, PARTIALLY,** (***sp+2; ​PC ← ​M[SP] are working*) whereas we couldn’t restore the flags for that it missed some branching functionalities.**

1. Hazard Detection Unit:

**It is just a matter of time, the HDU was prepared and compiled, except that we didn’t had the time to integrate it with the other processor components.**

1. Interrupt:

**With no excuses, we just couldn’t finish this instruction due to its complexity software wise.**

1. Forwarding Unit.

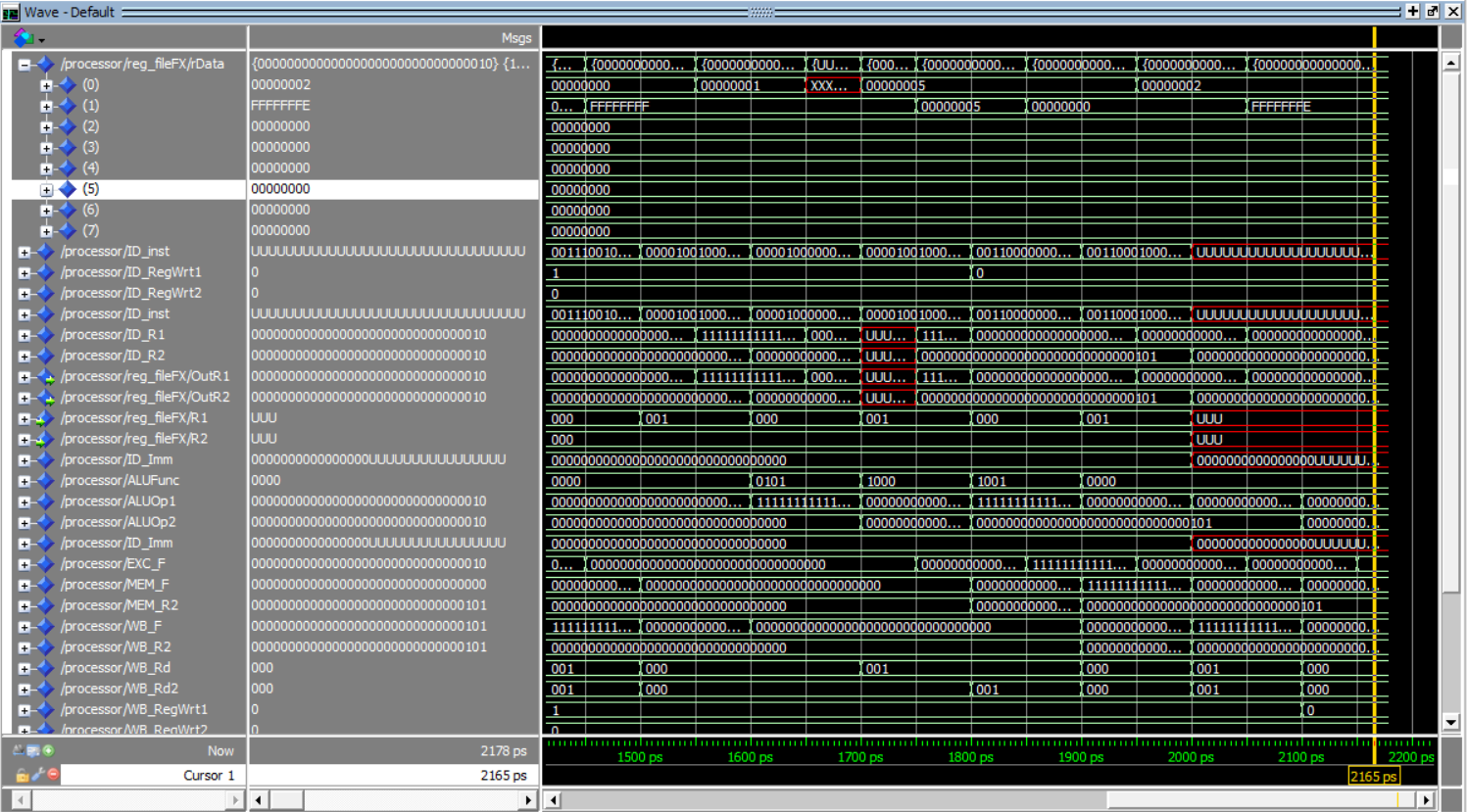
**Detailed Analysis**

1. The processor was tested without the forwarding unit and the branching detection unit, mainly with the one and two operands test cases.

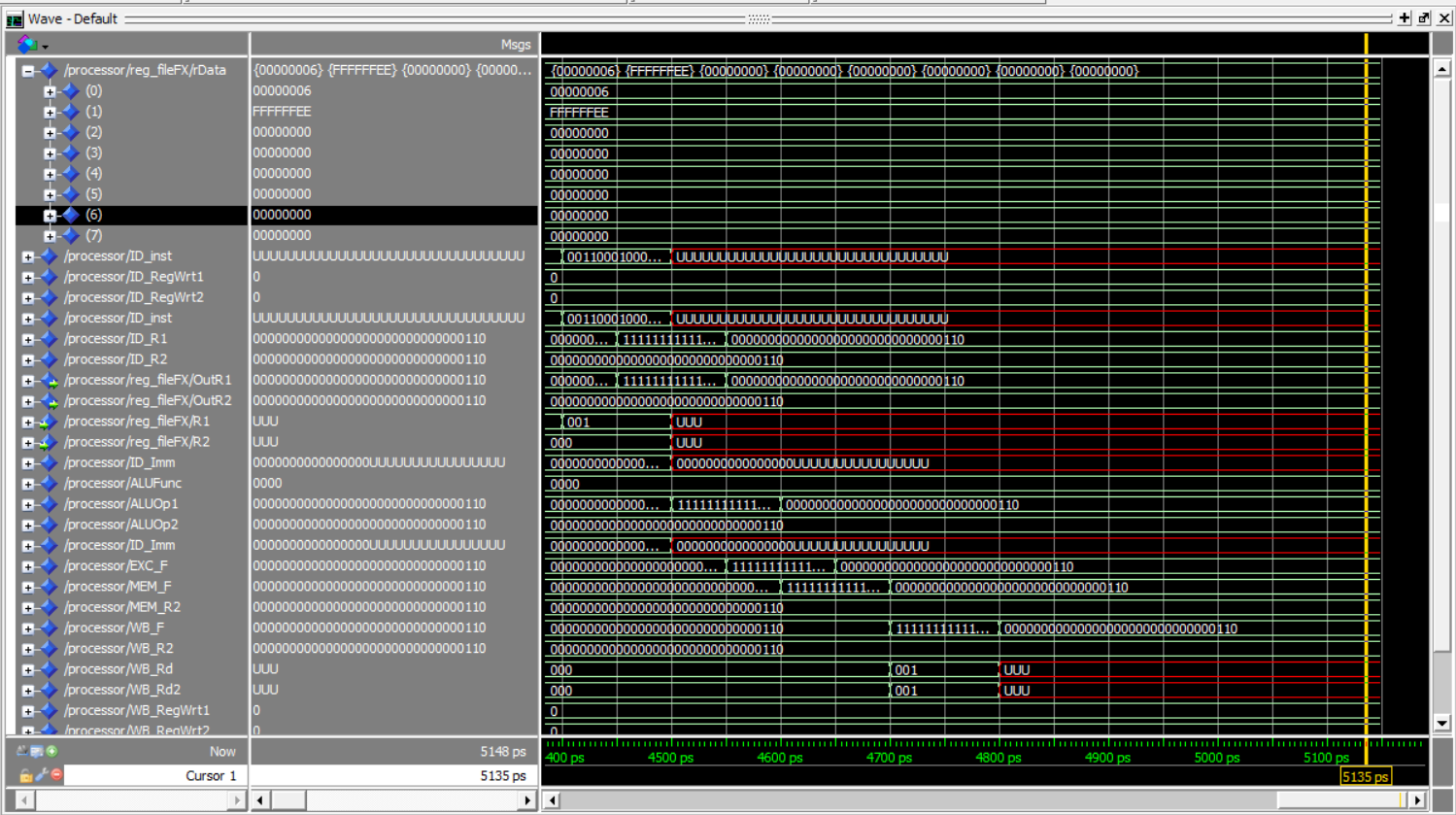
**Hazard**: Data Hazards. Specially the read after write for memory and register and load use hazards.

**Cases**: In line 15 & 16 in the one operand test cases the “R1” register encountered a read after write case due to a 2 consecutive ALU instruction. Consequently, the register, as shown in the fig, had provided the wrong data for the second instruction and its value was overwritten shortly after 3 cycles.

**One Operand Test With no NOPs**

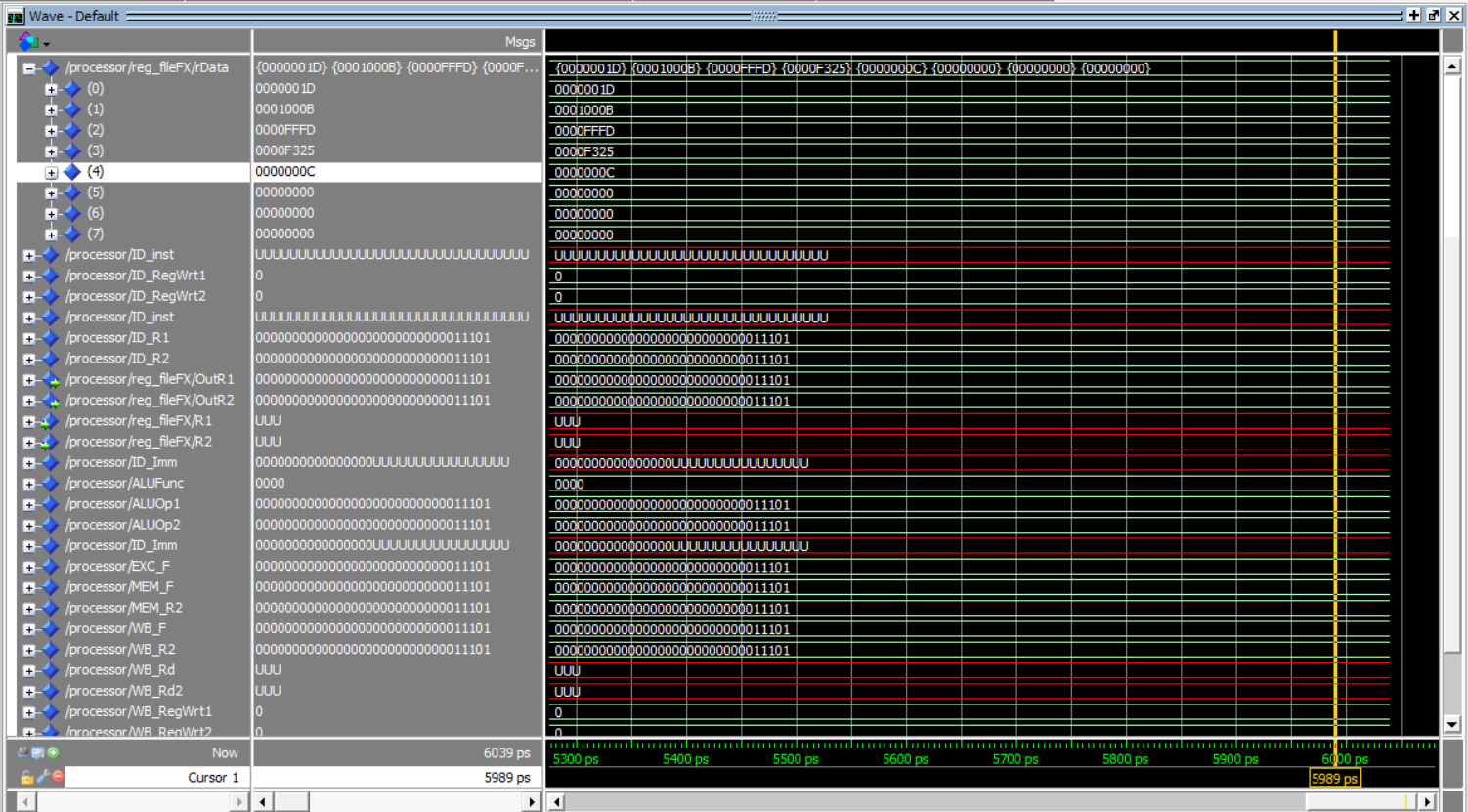
****

**One Operand Test With NOPs:**



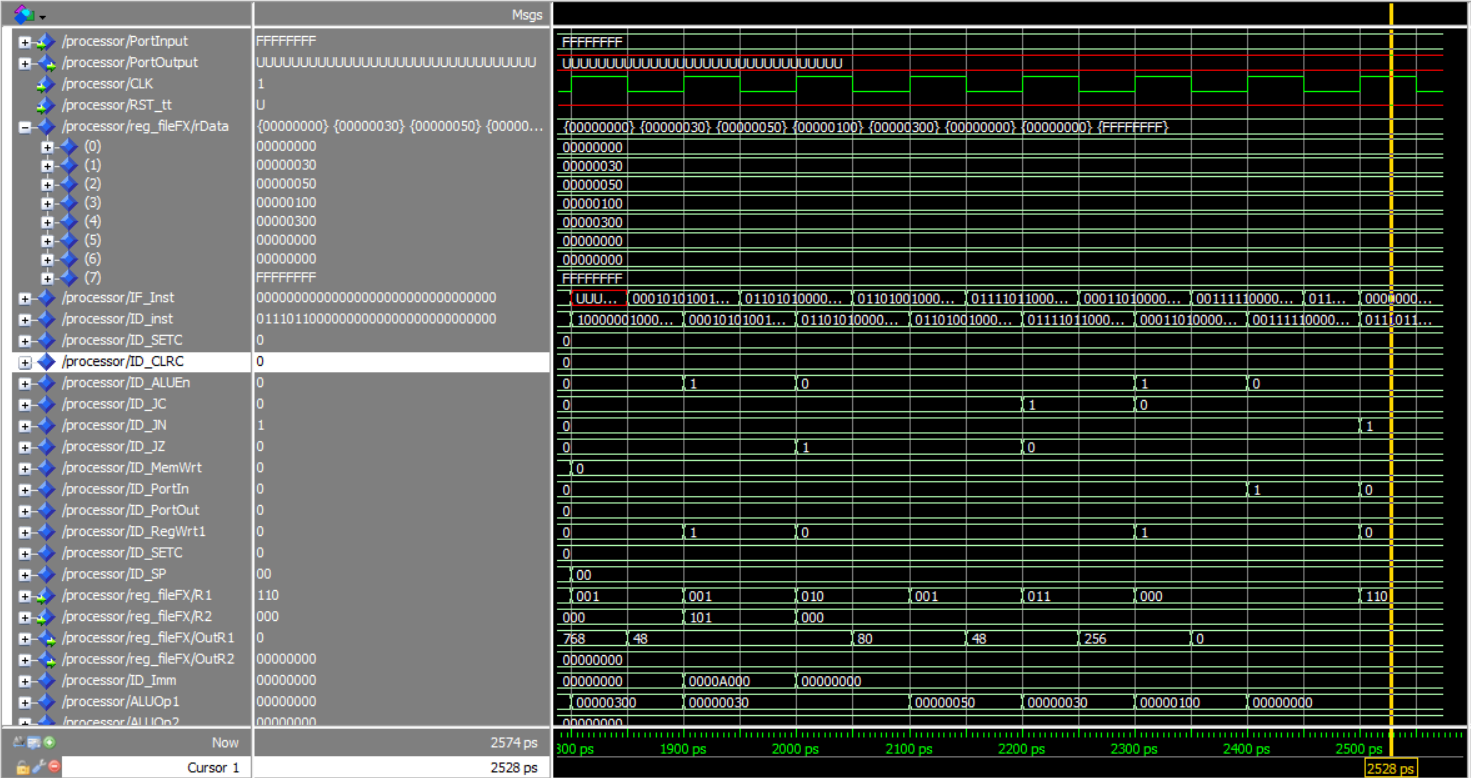
Snapshot of the one-operand test after execution after adding the NOP (registers are shifted: R1 🡪R0 and so forth)

**Two-operand Test**

****

Snapshot of the 2 operands test after execution with NOPs (registers are shifted: R1 🡪R0 and so forth)

**Branching Test**

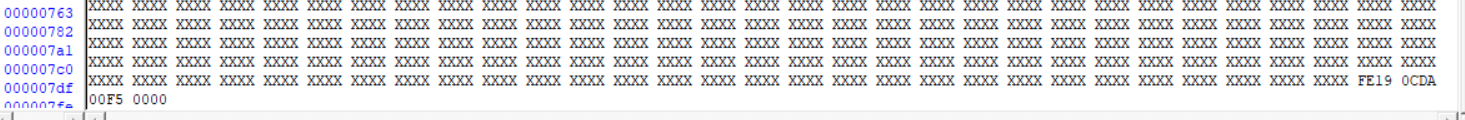
****

Branching Test Snapshot after adding a NOPs

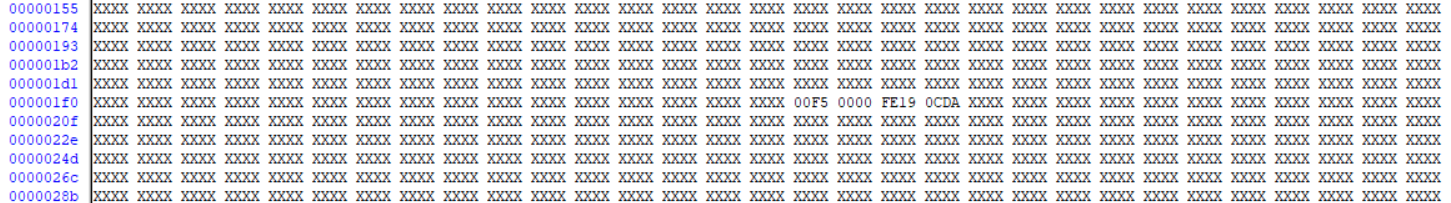
**Note about the Branching:**

In branch execution and predication, I succeeded in implementing it was 0 penalty cycles, though the principle used wasn’t 100% efficient, but the key idea was that the PC is generating the address in the second half cycle so that it calculates the address in the first half. In addition to, some auxiliary signals to achieve this performance like RC & RZ & RN. (i.e. the branching was detected and executed in the same cycle and no flushing was needed.) On the other hand not all test cases was implemented because of the RTI instruction.

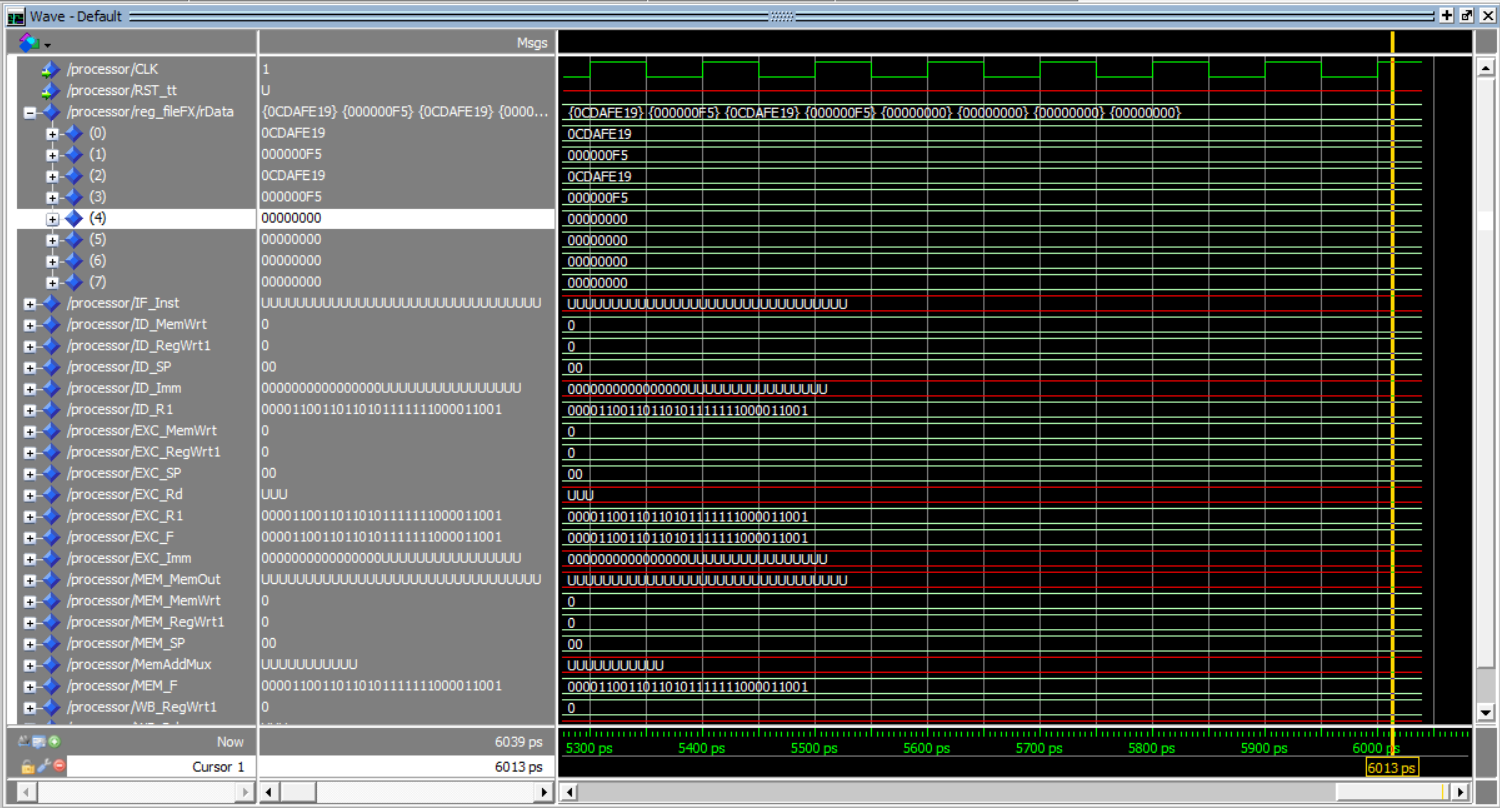
Memory Test



Stack snapshot after memory test



M[200:203] snapshot after memory test



Registers and Signals Snapshot after memory test