REPORT

In the part 1 implementation, for every load or store instruction, we had to access every time to the data memory to fetch corresponding data according to the current instruction. So for a latency of 5 CPU clock cycles we had to stall the CPU.

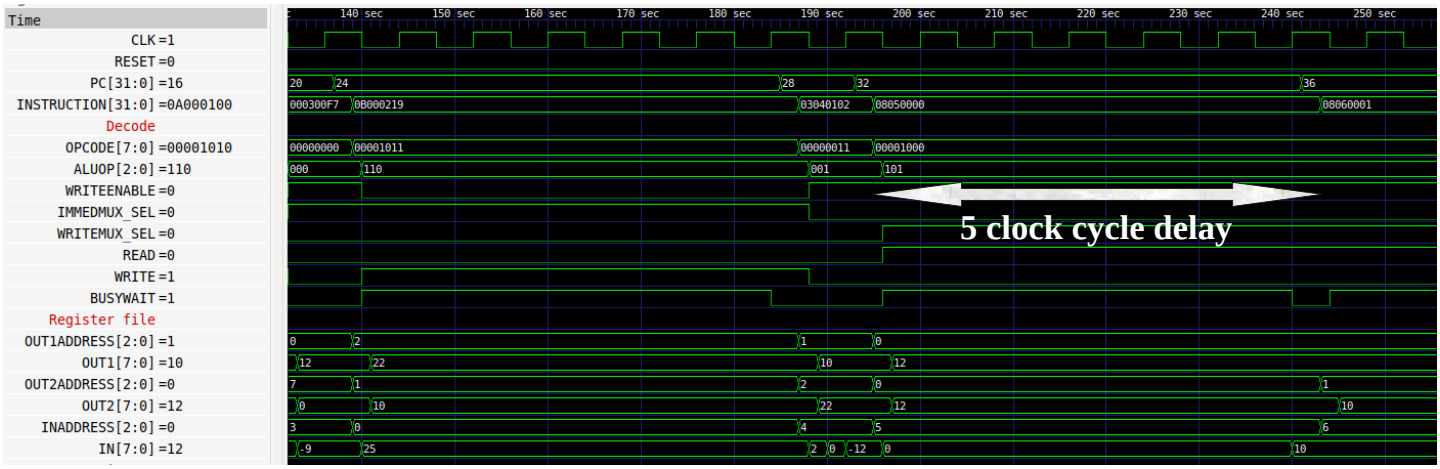

Figure 1: From part 1 implementation

- From PC = 32 to PC =36 there is a 5 clock cycle delay. Even the same instruction was fetched previously in the program CPU has fetched the data again from the data memory. (time consuming)

In real scenario it is most likely that the fetched instruction or data near to that instruction might be fetched soon to the CPU. Which is explained by the principles of locality.

So in order to improve memory access performance a cache memory can be used which works in between the CPU and data memory. Accessing data memory from CPU every time is a waste of clock cycles. But when a cache memory is used to store the once fetched data from the memory, the next time CPU needs those data it can be fetched without a clock cycle delay.

Therefore in the part 2 implementation, added a cache memory of size 32 bytes. Which is able to store 8 data blocks(block = 4 bytes contain 4 memory addresses and data). The CPU can access them using direct_mapped block placement. So when a memory request occurs first the CPU checks if the required data is available in the cache and if so fetch fast and the CPU continues without stalling. (no clock cycles lost)

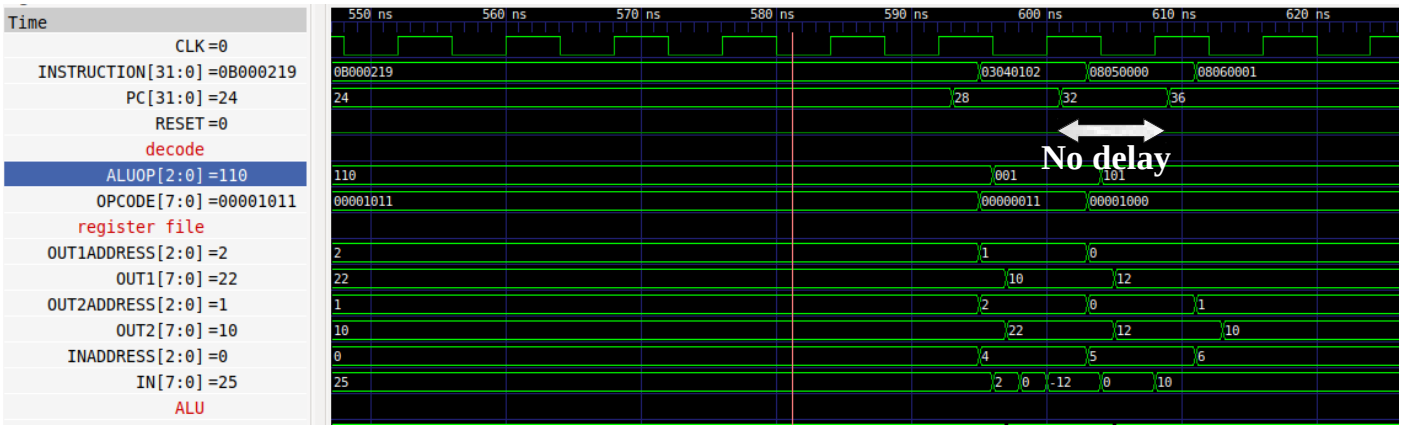

Figure 2: From part 2 implementation

- From PC =32 to PC=36 there is no stalling of the CPU. Since the instruction at PC = 32 was fetched before and is stored in the cache, CPU can access it fast and run without stalling.