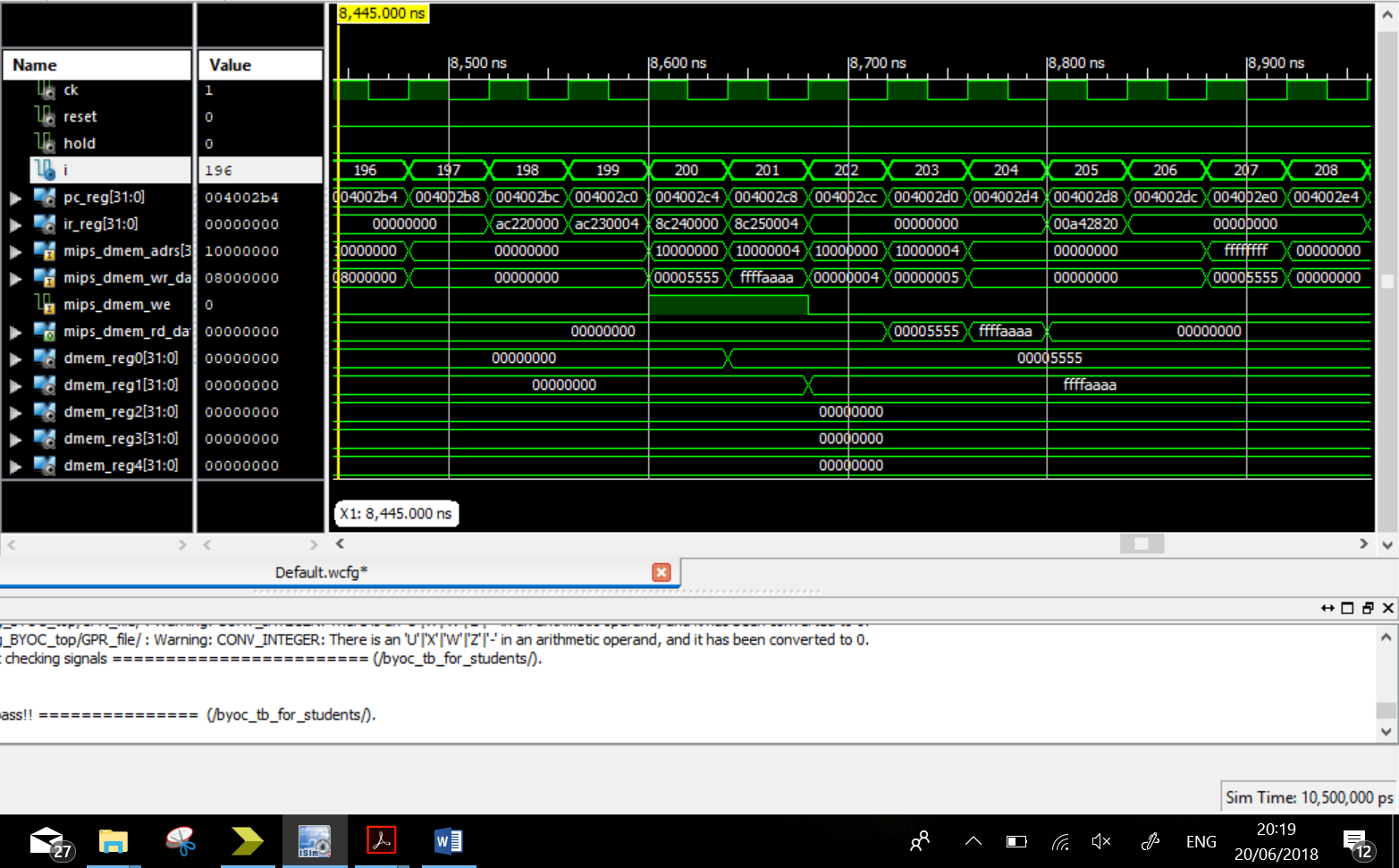
Simulation report

MIPS CPU

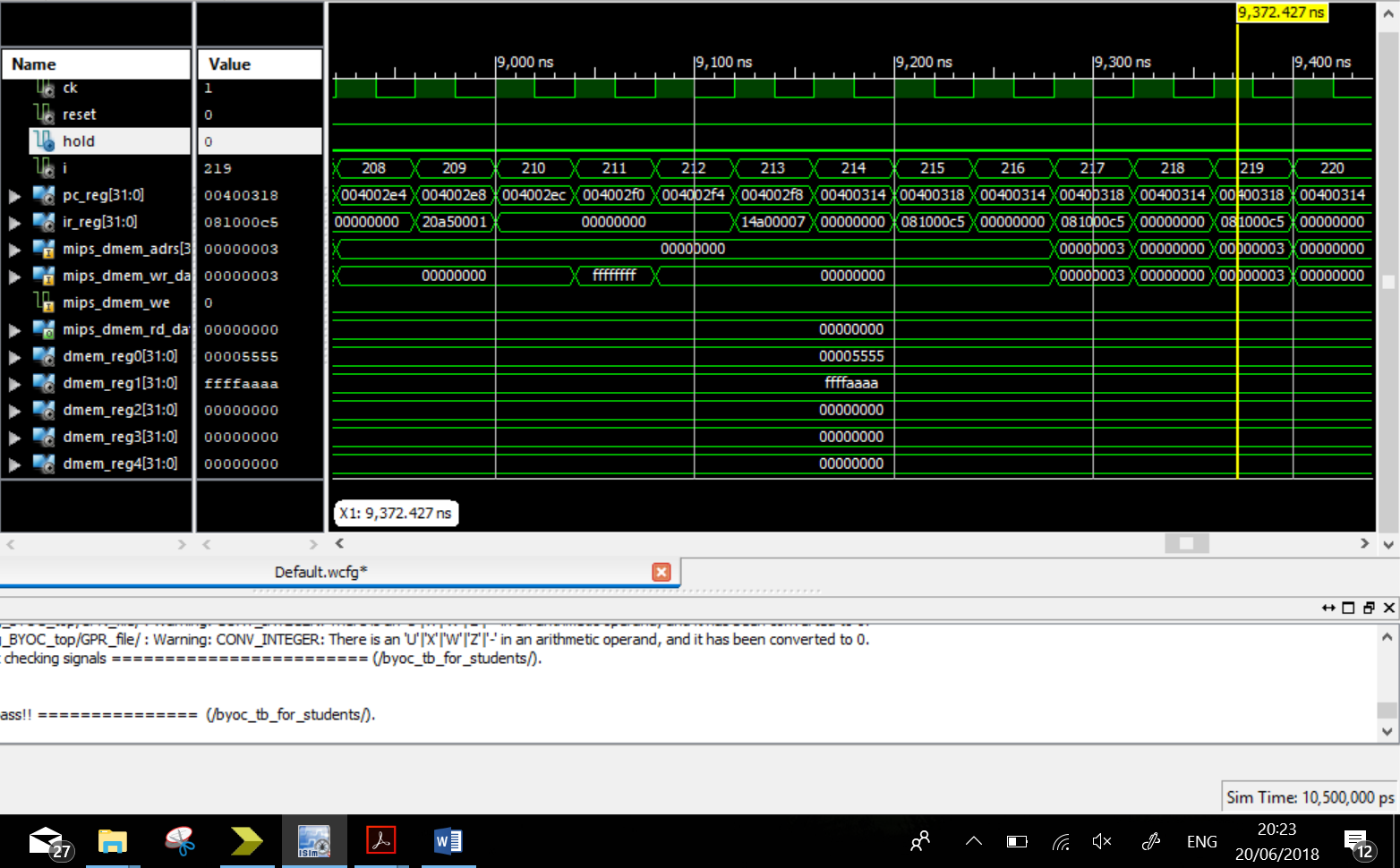
Lidor Cohen 305782732

Otabek Sherman 326868775

Question 3.1



Clocks 196-208



Clocks 208-220

Question 3.2

In the cycles 196 – 220 we can see the running of the R-type and J-Type commands and also testing of the of the new I-type commands that we added - lw and sw.

|  |  |  |  |
| --- | --- | --- | --- |
| CK | Instruction | Meaning | Comment |
| 197-200 | Sw $2, 0($1) | Memory(GPR[$1]+0=GPR[$2] | Storing the data in memory |
| 198-201 | Sw $3, 4($1) | Memory(GPR[$1] + 4 = GPR[$3]) | Storing the data in the memory |
| 199-203 | Lw $4, 0($1) | GPR[$4]=Memory(GPR[$1] + 0) | Loading data from the memory |
| 200-204 | Lw $5, 4($1) | GPR[$5]=Memory(GPR[$1] + 4) | Loading data from the memory |
| 201-207 | nop |  |  |
| 208-211 | Addi $5, $5, 1 | GPR[$5] = Memory(GPR[$5]+1) | Adding the scalar |
| 212-215 | Bne $5, $0, 0x0007 | If Rs != Rt, PC = PC + 4 + sext(imm)\*4;  Else PC = PC + 4; | Jumping the address if the condition exists |
| 216-218 | J 0x00c1 | PC = 0x00c1\*4 | Jumping address |

Question 3.3

Should be inserted 3 nop instructions between two consecutive R-Type instructions if the 2nd one uses the result of the 1st one. The R-Type instruction requires at least 3 clock cycles between them if the 2nd instruction uses the result of the 1st one because while writing the data to registers it must to go through the MIPS phases, one additional cycle in comparison to the previous exercise, since we have added the MEM phase.

Question 3.4

Translation of the code:

Addi $3, $0, 0 //set i=0

Addi $1, $0, 10 //set r5=10

Nop

Loop:

Nop

Nop

Beq $3, $1, continue

Nop

{Loop}

Addi $3, $3, 1

J Loop

Continue:

The R-Type instruction must wait 3 nop instructions between two consecutive R-Type instructions if the 2nd one uses the result of the 1st one. After the command is fetched, decoded and executed, the jump instruction happens after 3 clock cycles. Also in beq instruction we must to compare between the Rs and Rt registers, that requires decoding, fetching and comparing. We must wait until the signal stating whether to increment the PC by 4 or Jump to an offset is received. We will see the result after the ALU subs 2 values that stored in the registers, in the end of the ID phase.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction |  |  |  |  |  |  |  |  |
| beq | IF | ID | EX | MEM | WB |  |  |  |
| nop |  | IF | ID | EX | MEM | WB |  |  |
| No branch |  |  |  | IF | ID | EX | MEM | WB |
| … |  |  |  |  |  |  |  |  |
| Branch |  |  |  | IF | ID | EX | MEM | WB |

Question 3.5

Possible SW & HW based solutions:

SW

* For lw, addi, bne instructions we can to use nops or to insert real operations instead of nop instructions.
* For branch instructions(bne, beq) we can to predict whether or not a branch will occur.

HW

* We can use 2 ways of forwarding: branch and data forwarding. By the branch forwarding we can save the nops latency in branch instructions. By data forwarding we can save the nops latency in R-Type instructions.