Laboratory Work 4 Report

ISA & Datapath Design for Multi-Cycle CPU

Mustafa BIYIK

2231454

The report consists of four main parts.

The first part is answers of the preliminary work part as hard-copy. Hardcopy contains ISA configurations and the design.

The second part is verification results of the changed modules in wvf platform of Quartus.

The third part is the simulation result of the experimental part. Testbench is applied on the Modelsim.

The last part contains the codes of the Datapath design and its testbench.

1. Preliminary Work Part(Hard-copy)

2. Verification of the modules

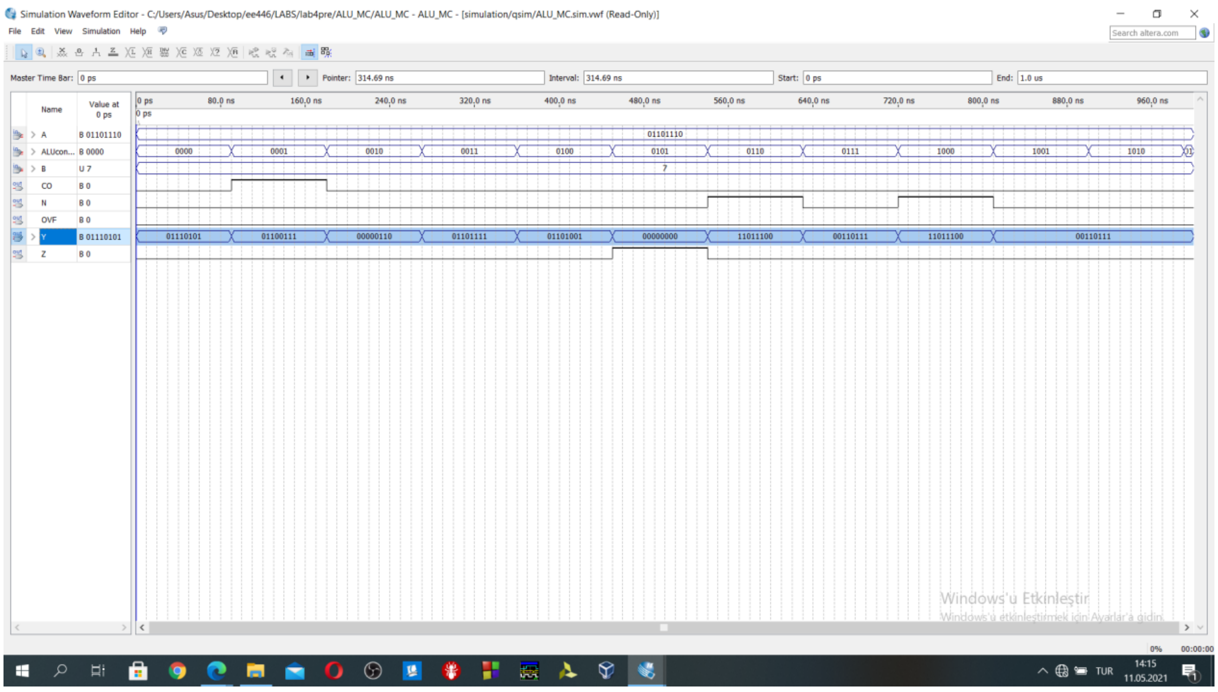


Figure 1: ALU simulation

In Figure 1, the simulation result of the ALU exists.

4'b0000: //add A (A)01101110+(B)00000111=01110101 matches with Y in the Figure 1(result)

4'b0001: //subt a-b (A)01101110-(B)00000111=01100111 matches with Y in the Figure 1(result)

4'b0010: //and (A)01101110&(B)00000111=00000110 matches with Y in the Figure 1(result)

4'b0011: //or (A)01101110|(B)00000111=01101111 matches with Y in the Figure 1(result)

4'b0100://xor (A)01101110+(B)00000111=01101001 matches with Y in the Figure 1(result)

4'b0101://clear clear then Y=0

4'b0110: //rol (A)01101110=11011100(Y) matches with Y in the Figure 1(result)

4'b0111://ror (A)01101110=00110111(Y) matches with Y in the Figure 1(result)

4'b1000: //shift left lsl (A)01101110=11011100(Y) matches with Y in the Figure 1(result)

4'b1001: //shift right lsr (A)01101110=00110111(Y) matches with Y in the Figure 1(result)

4'b1010 //arithmetic shift right (A)01101110=00110111 (Y) matches with Y in the Figure 1(result)

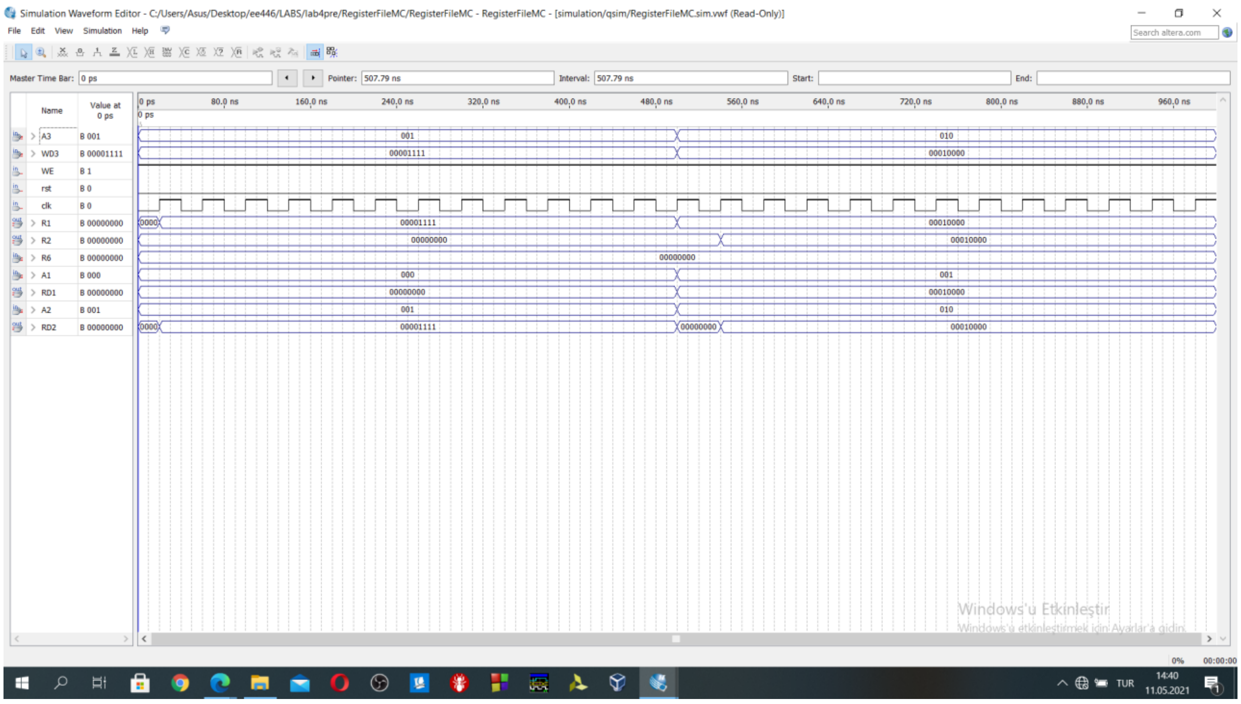

Simulation Waveform Editor - C:/Users/Asus/Desktop/ee446/LABS/lab4pre/RegisterFileMC/RegisterFileMC - RegisterFileMC - [simulation/qsim/RegisterFileMC.sim.vwf (Read-Only)]
Master Time Bar: 0 ps
Pointer: 507.79 ns
Interval: 507.79 ns
A3 B 001 — 001, 010
WD3 B 00001111 — 00001111, 00010000
WE B 1
rst B 0
clk B 0
R1 B 00000000 — 0000, 00001111, 00010000
R2 B 00000000 — 00000000, 00010000
R6 B 00000000 — 00000000
A1 B 000 — 000, 001
RD1 B 00000000 — 00000000, 00010000
A2 B 001 — 001, 010
RD2 B 00000000 — 0000, 00001111, 00000000, 00010000

Figure 2:Register File Simulation

In Figure 2, register File for the Datapath design is designed. R1 and R2 are for the demonstration purposes. R6 is PC register. Firstly, R1(with A3) is loaded with 00001111 then the content of the R1 can be seen at the “R1” output. Also, when the A2=1, then RD2 value contains the current content of the R1 after the write operation of the R1. Then, both R1 and R2 is loaded with the 00010000 value, their value can be observed from the “R1” and “R2” outputs. They are also can be seen RD1 and RD2 when the A1 and A2 is 1 or 2.

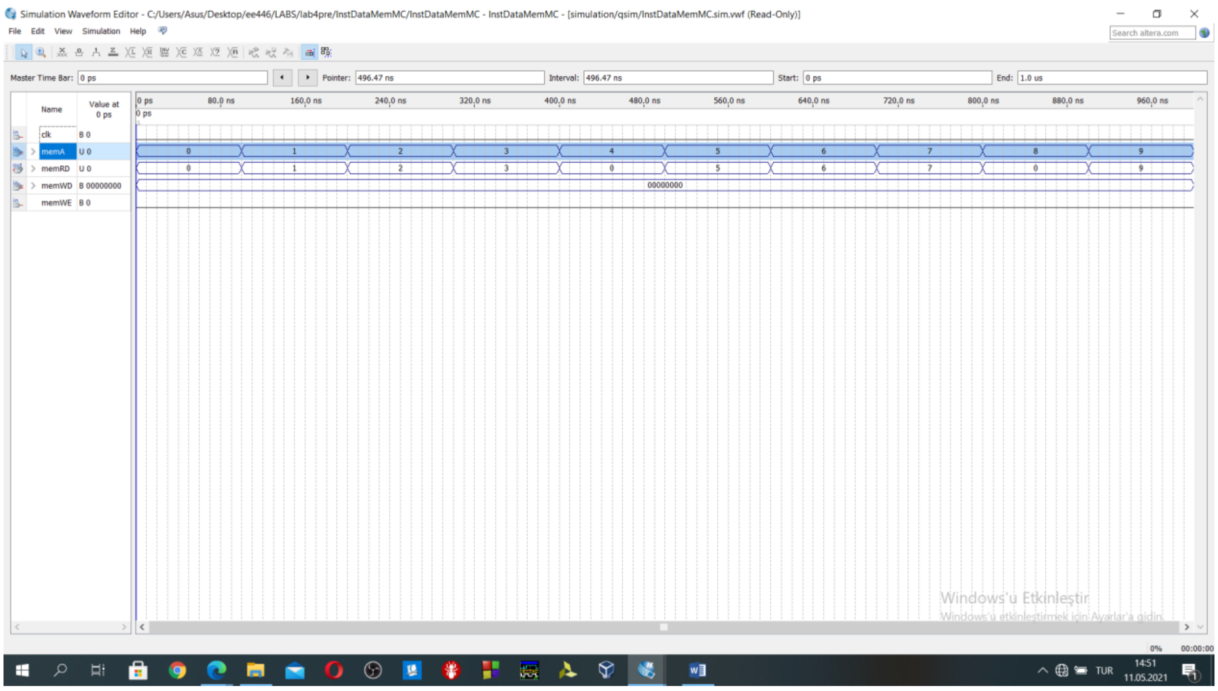

Simulation Waveform Editor - C:/Users/Asus/Desktop/ee446/LABS/lab4pre/InstDataMemMC/InstDataMemMC - InstDataMemMC - [simulation/qsim/InstDataMemMC.sim.vwf (Read-Only)]
Master Time Bar: 0 ps
Pointer: 496.47 ns
Interval: 496.47 ns
Start: 0 ps
End: 1.0 us
clk B 0
memA U 0 — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
memRD U 0 — 0, 1, 2, 3, 0, 5, 6, 7, 0, 9
memWD B 00000000 — 00000000
memWE B 0

Figure 3:memory read simulation

In Figure 3, memA is memory address, memRD contains the memory read output which is pointed by memA. While initializing the memory, 0,4,8… contains the instructions. 1,2,3,5,6,7,9,10,11… contains the data which is equal to the address no. Initially memory does not contain any instruction. Therefore, all of them are equal to 0.

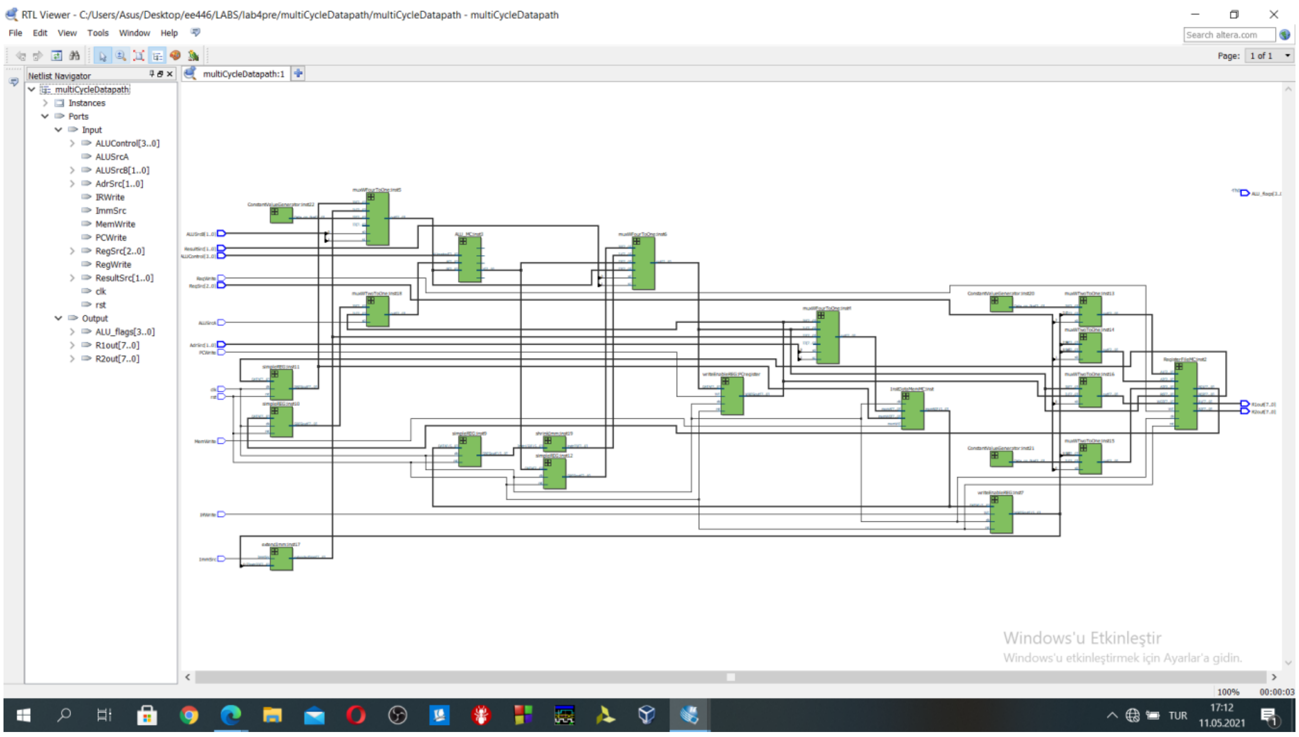
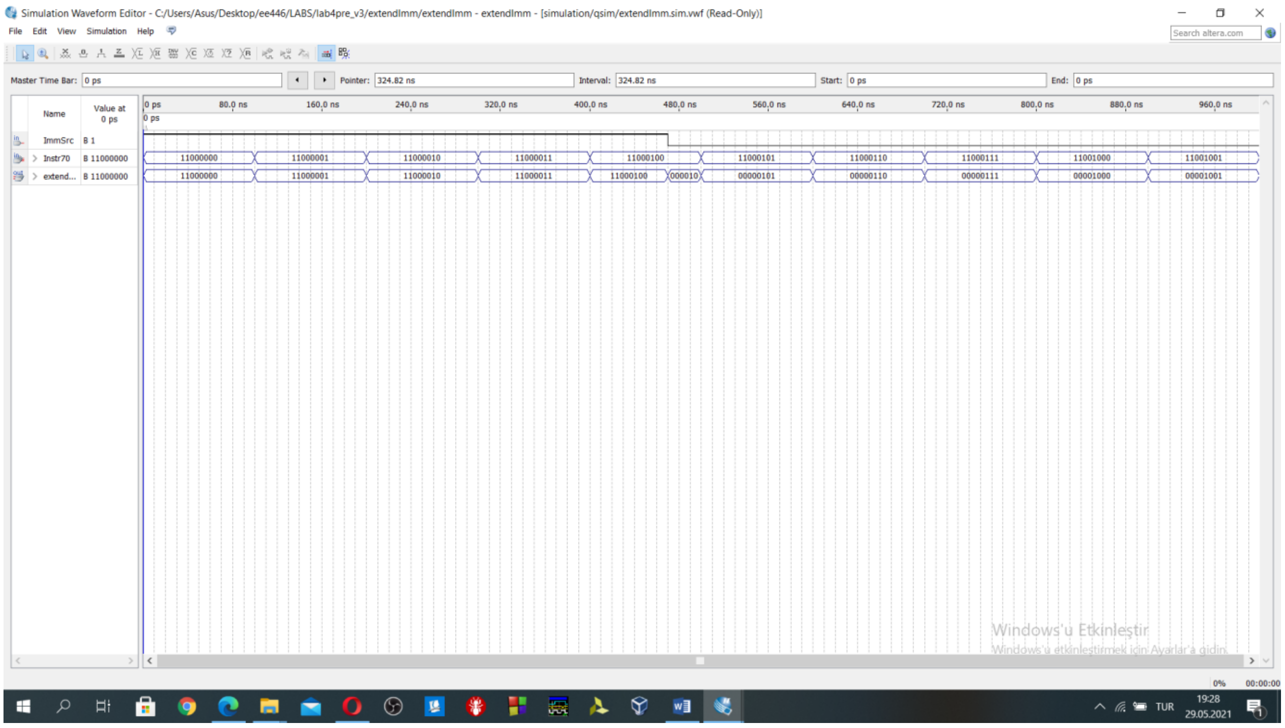


Figure 4:netlist view of the multiCycleDatapath

Figure 4 contains the netlist of the design.


Simulation Waveform Editor - C:/Users/Asus/Desktop/ee446/LABS/lab4pre_v3/extendImm/extendImm - extendImm - [simulation/qsim/extendImm.sim.vwf (Read-Only)]
File Edit View Simulation Help

Master Time Bar: 0 ps   Pointer: 324.82 ns   Interval: 324.82 ns   Start: 0 ps   End: 0 ps

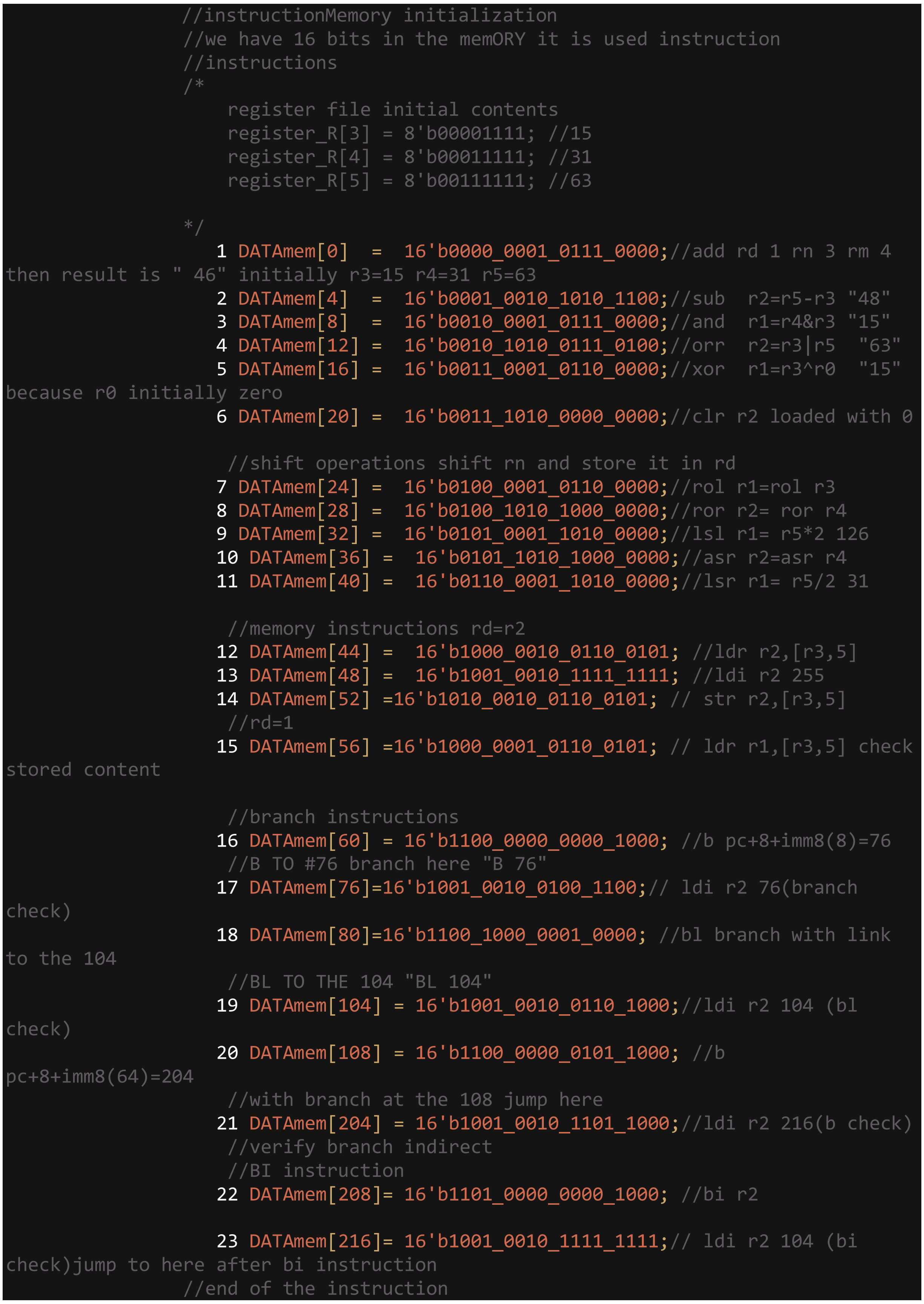
```
Name       Value at 0 ps   0 ps      80.0 ns   160.0 ns  240.0 ns  320.0 ns  400.0 ns  480.0 ns  560.0 ns  640.0 ns  720.0 ns  800.0 ns  880.0 ns  960.0 ns
ImmSrc     B 1
Instr70    B 11000000      11000000  11000001  11000010  11000011  11000100  11000101  11000110  11000111  11001000  11001001
extend...  B 11000000      11000000  11000001  11000010  11000011  11000100  000010  00000101  00000110  00000111  00001000  00001001
```

Figure 5: ExtendImm simulation result

When the ImmSrc=1, then input is directly served as output. When the ImmSrc=0, the last five bits of the input are extended as 8 bits by adding additional 0 to the first five bits of the input.

3. Simulation result of the Experimental work via Testbench

add-sub-and-orr-xor-clr-rol-ror-lsl-asr-lsr-ldr-ldi-str-b-bl-bi instructions in memory



```
                //instructionMemory initialization
                //we have 16 bits in the memORY it is used instruction
                //instructions
                /*
                    register file initial contents
                    register_R[3] = 8'b00001111; //15
                    register_R[4] = 8'b00011111; //31
                    register_R[5] = 8'b00111111; //63

                */
                   1 DATAmem[0]  =  16'b0000_0001_0111_0000;//add rd 1 rn 3 rm 4
then result is " 46" initially r3=15 r4=31 r5=63
                   2 DATAmem[4]  =  16'b0001_0010_1010_1100;//sub  r2=r5-r3 "48"
                   3 DATAmem[8]  =  16'b0010_0001_0111_0000;//and  r1=r4&r3 "15"
                   4 DATAmem[12] =  16'b0010_1010_0111_0100;//orr  r2=r3|r5  "63"
                   5 DATAmem[16] =  16'b0011_0001_0110_0000;//xor  r1=r3^r0  "15"
because r0 initially zero
                   6 DATAmem[20] =  16'b0011_1010_0000_0000;//clr r2 loaded with 0

                    //shift operations shift rn and store it in rd
                   7 DATAmem[24] =  16'b0100_0001_0110_0000;//rol r1=rol r3
                   8 DATAmem[28] =  16'b0100_1010_1000_0000;//ror r2= ror r4
                   9 DATAmem[32] =  16'b0101_0001_1010_0000;//lsl r1= r5*2 126
                   10 DATAmem[36] =  16'b0101_1010_1000_0000;//asr r2=asr r4
                   11 DATAmem[40] =  16'b0110_0001_1010_0000;//lsr r1= r5/2 31

                    //memory instructions rd=r2
                   12 DATAmem[44] =  16'b1000_0010_0110_0101; //ldr r2,[r3,5]
                   13 DATAmem[48] =  16'b1001_0010_1111_1111; //ldi r2 255
                   14 DATAmem[52] =16'b1010_0010_0110_0101; // str r2,[r3,5]
                    //rd=1
                   15 DATAmem[56] =16'b1000_0001_0110_0101; // ldr r1,[r3,5] check
stored content

                    //branch instructions
                   16 DATAmem[60] = 16'b1100_0000_0000_1000; //b pc+8+imm8(8)=76
                    //B TO #76 branch here "B 76"
                   17 DATAmem[76]=16'b1001_0010_0100_1100;// ldi r2 76(branch
check)

                   18 DATAmem[80]=16'b1100_1000_0001_0000; //bl branch with link
to the 104
                    //BL TO THE 104 "BL 104"
                   19 DATAmem[104] = 16'b1001_0010_0110_1000;//ldi r2 104 (bl
check)

                   20 DATAmem[108] = 16'b1100_0000_0101_1000; //b
pc+8+imm8(64)=204
                    //with branch at the 108 jump here
                   21 DATAmem[204] = 16'b1001_0010_1101_1000;//ldi r2 216(b check)
                    //verify branch indirect
                    //BI instruction
                   22 DATAmem[208]= 16'b1101_0000_0000_1000; //bi r2

                   23 DATAmem[216]= 16'b1001_0010_1111_1111;// ldi r2 104 (bi
check)jump to here after bi instruction
                //end of the instruction
```

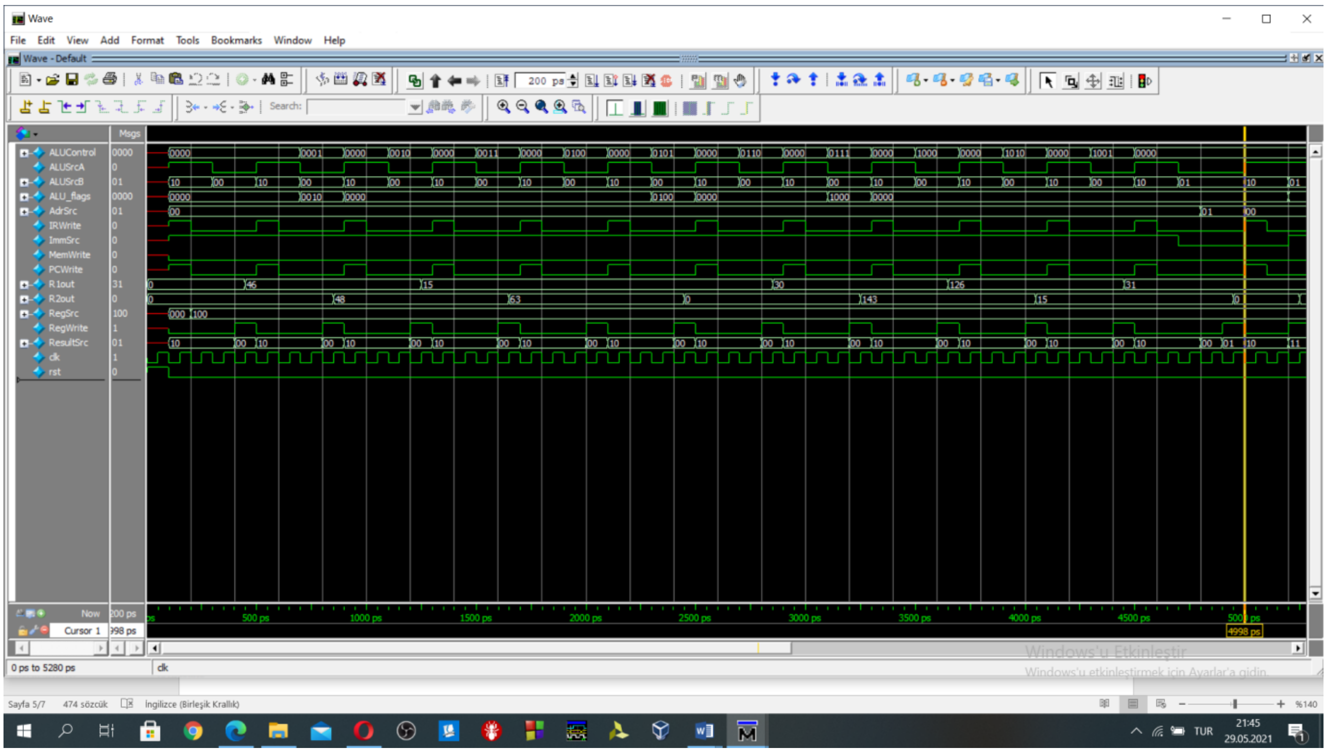


Figure 6: The simulation result between 0-5000ps

As in the datamem, the first executed instruction address is 0 because PC counter value is initially. It’s value increases four by four as long as PC counter does not run into branch instruction.

R1out & R2out value are used as demonstration purposes. They provide r1 and r2 values in the register file.

<u>1st instruction ADD</u>(four cycles)(100ps-500ps) As in the Figure 6, R1out output value is updated as 46 after four clock cycle. r1=r3+r4 ➔ r1=15+31=46, calculated value and simulation value are matching.

<u>2nd instruction SUB</u>(four cycles)(500ps-900ps) As in the Figure 6, R2out output value is updated as 48 after four clock cycle. r2=r5-r3 ➔ r2=63-15=48, calculated value and simulation value are matching.

<u>3rd instruction AND</u>(four cycles)(900ps-1300ps) As in the Figure 6, R1out output value is updated as (15)1111 after four clock cycle. r1=r3&r4 ➔ r1=00001111(15)&00011111(31)=00001111(15), calculated value and simulation value are matching.

<u>4th instruction ORR</u>(four cycles)(1300ps-1700ps) As in the Figure 6, R2out output value is updated as (63)111111 after four clock cycle. r2=r3&r5 ➔ r2=00001111(15)&00111111(63)=00111111(63), calculated value and simulation value are matching.

<u>5th instruction XOR</u>(four cycles)(1700ps-2100ps) As in the Figure 6, R1out output value is updated as (15)1111 after four clock cycle. r1=r3^r0 ➔ r1=00001111(15)^00000000(0)=00001111(15), calculated value and simulation value are matching.

<u>6th instruction CLR</u>(four cycles)(2100ps-2500ps) As in the Figure 6, R2out output value is updated as 0 after four clock cycle. r2←0, ➔ r2=00000000(0) calculated value and simulation value are matching.

<u>7th instruction ROL</u>(four cycles)(2500ps-2900ps) As in the Figure 6, R1out output value is updated as (30)00011110 after four clock cycle. r1=r3(00001111) rotate left by 1 ➔ r1=00011110(30), calculated value and simulation value are matching.

<u>8th instruction ROR</u>(four cycles)(2900ps-3300ps) As in the Figure 6, R2out output value is updated as (143)10001111 after four clock cycle. r2=r3(00001111) rotate right by 1 ➔ r2=10001111(143), calculated value and simulation value are matching.

<u>9th instruction LSL</u>(four cycles)(3300ps-3700ps) As in the Figure 6, R1out output value is updated as (126)01111110 after four clock cycle. r1=r3(00001111) << 1 ➔ r1=01111110(126), calculated value and simulation value are matching.

<u>10th instruction ASR</u>(four cycles)(3700ps-4100ps) As in the Figure 6, R2out output value is updated as (15)00001111 after four clock cycle. r2=r4(00011111) arithmetic shift right by 1 ➔ r2=00001111(15), calculated value and simulation value are matching.

<u>11th instruction LSR</u>(four cycles)(4100ps-4500ps) As in the Figure 6, R1out output value is updated as (31)00011111 after four clock cycle. r1=r5(00111111) >> 1 ➔ r1=00011111(31), calculated value and simulation value are matching.

<u>12th instruction LDR</u>(five cycles)(4500ps-5000ps) As in the Figure 6, R2out output value is updated as 0 after five clock cycle. ldr r2,[r3,5] ➔ r2 is loaded with data at the memory address 20. The data is equal to 0 at pointed by the address. Calculated value and simulation value are matching. Why it is 0, because imm8 is 0000 0000.

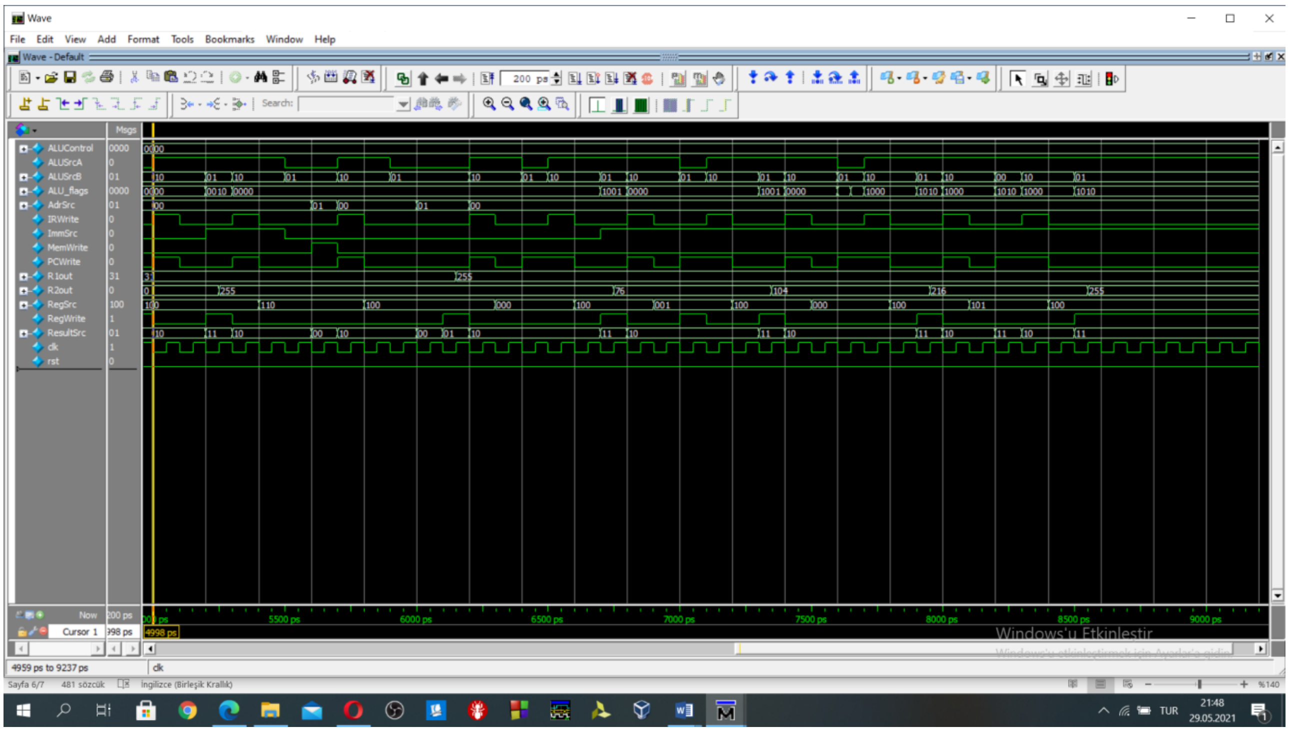


Figure 7: The simulation result between 5000ps-9000ps

<u>13th instruction LDI</u>(three cycles)(5000ps-5300ps) As in the Figure 7, R2out output value is updated as 255(1111 1111) after three clock cycle. imm8 value is moved to the r2 register. The moved value can be observed from Figure 7. Calculated value and simulation value are matching.

<u>14th instruction STR</u>(four cycles)(5300ps-5700ps) As in the Figure 7, str r2,[r3,5] ➔ r2 is stored to the "[r3]+5" address as 0000 0000 1111 1111.

//checking STR with LDR

<u>15th instruction LDR</u> (five cycles) (5700ps-6200ps) As in the Figure 7, R1out output value is updated as 255 after five clock cycle. ldr r1,[r3,5] ➔ r1 is loaded with data at the memory address 20. The data is equal to 255 at pointed by the address. Calculated value and simulation value are matching.

<u>16th instruction B(branch)</u>(three cycles)(6200ps-6500ps). When we look at the instruction memory, imm8 bits of the branch are 0000 1000(8), the next executed instruction must be PC+8+imm8. Current PC counter value is equal to 60. The next instruction must be at the address value 76.

Note: In my design, I do not multiply immediate with 4.

//check Branch working or not,

//imm8 value is equal to branched PC counter value. Then it's value is moved to the r2 as 76.

<u>17th instruction LDI</u>(three cycles)(6500ps-6800ps) As in the Figure 7, R2out output value is updated as 76(0100 1100) after three clock cycle. imm8 value is moved to the r2 register. The moved value can be observed from Figure 7. Calculated value and simulation value are matching.

18th instruction BL(branch with link)(three cycles)(6800ps-7100ps). When we look at the instruction memory, imm8 bits of the branch are 0001 0000(16), the next executed instruction must be PC+8+imm8. Current PC counter value is equal to 80. The next instruction must be at the address value 104.

19th instruction LDI(three cycles)(7100ps-7400ps) As in the Figure 7, R2out output value is updated as 104(0110 1000) after three clock cycle. imm8 value is moved to the r2 register. The moved value can be observed from Figure 7. Calculated value and simulation value are matching.

20th instruction B(branch)(three cycles)(7400ps-7700ps). When we look at the instruction memory, imm8 bits of the branch are 0101 1000(88), the next executed instruction must be PC+8+imm8. Current PC counter value is equal to 108. The next instruction must be at the address value 204.

21th instruction LDI(three cycles)(7700ps-8000ps) As in the Figure 7, R2out output value is updated as 216(1101 1000) after three clock cycle. imm8 value is moved to the r2 register. The moved value can be observed from Figure 7 as 216. Calculated value and simulation value are matching.

//R2=216, now I demonstrate branch indirect

22th instruction BI(branch indirect)(three cycles)(8000ps-8300ps). When we look at the instruction memory, rm=r2=216. The next executed instruction must be [r2] address. Current PC counter value is equal to 208. The next instruction must be at the address value 216 because r2's value is equal to 216.

23th instruction LDI(three cycles)(8300ps-8600ps) As in the Figure 7, R2out output value is updated as 255(1111 1111) after three clock cycle. imm8 value is moved to the r2 register. The moved value can be observed from Figure 7 as 255. Calculated value and simulation value are matching.

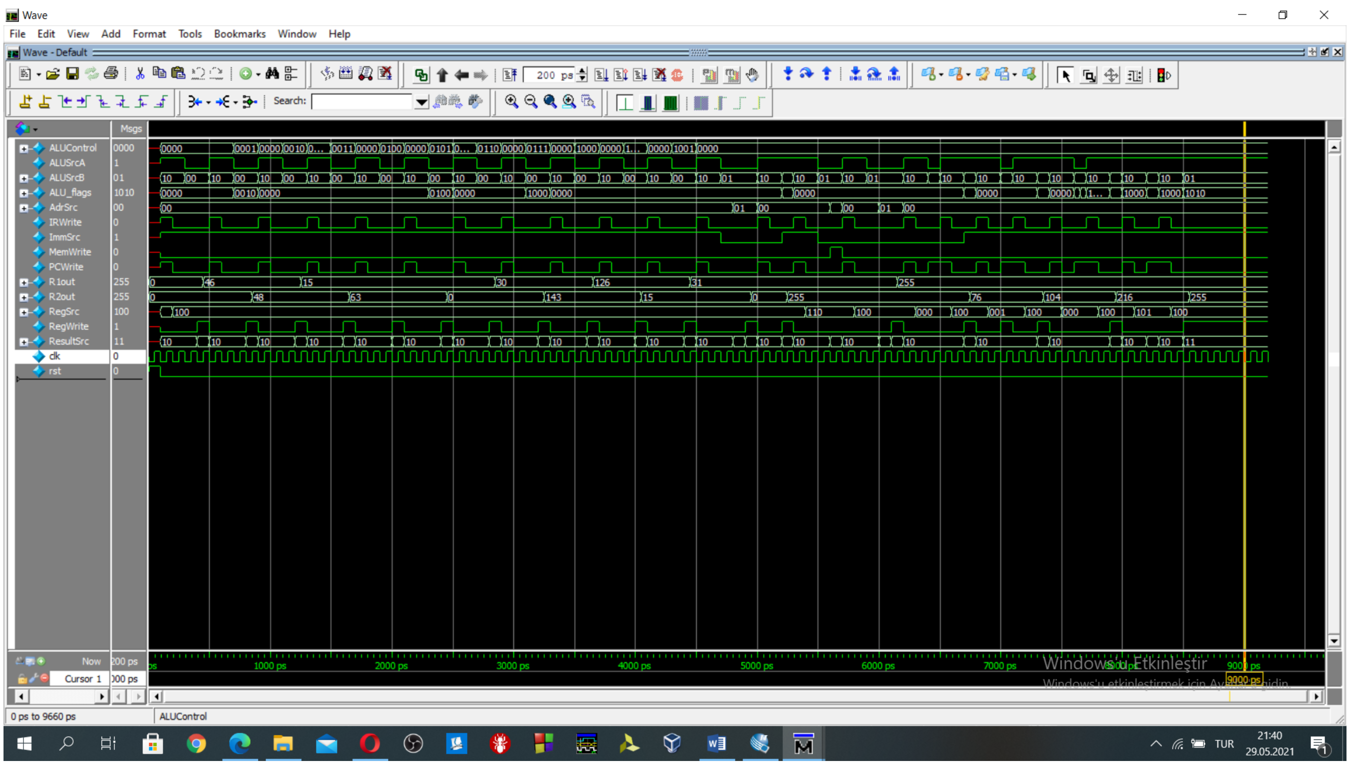
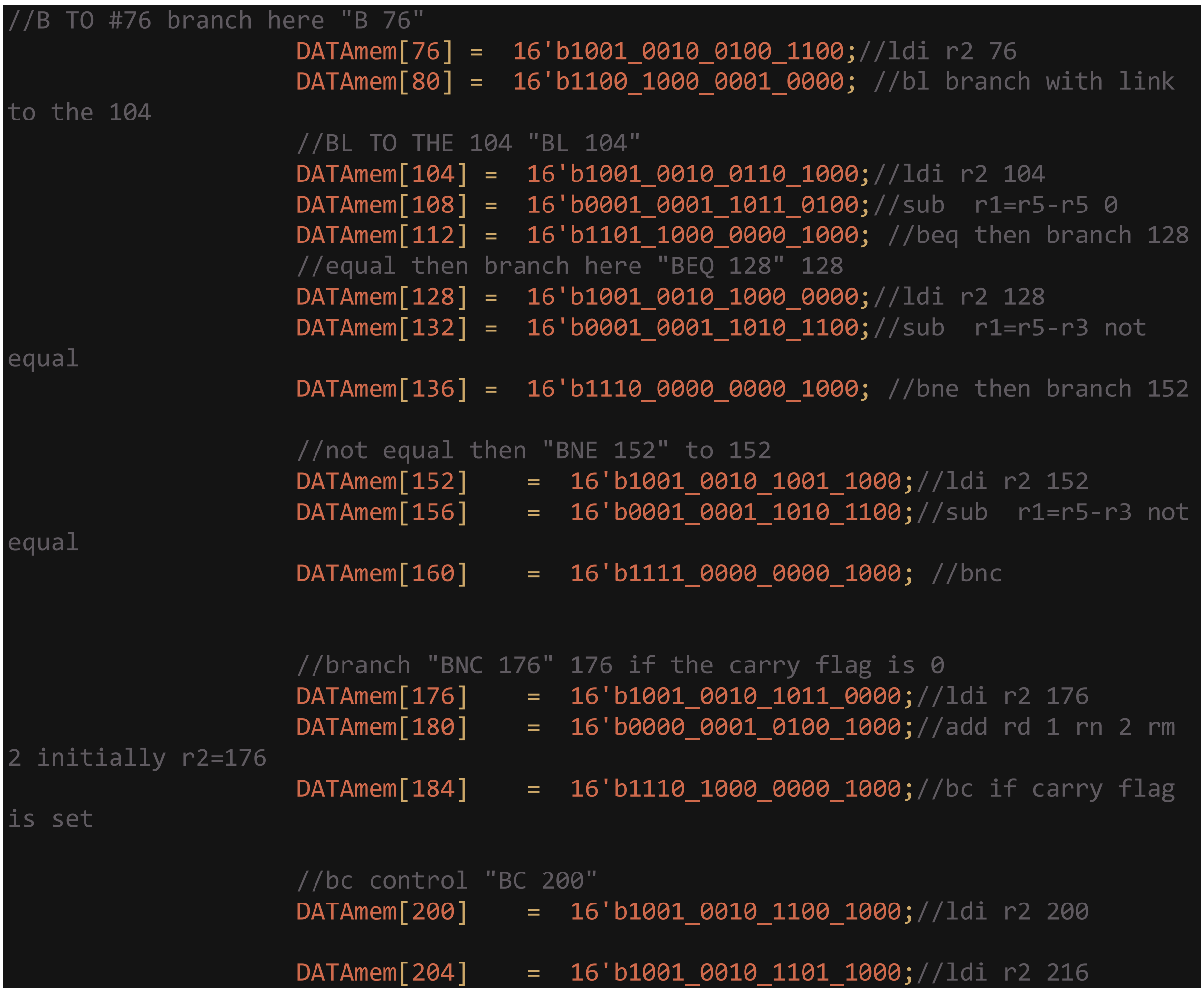


Figure 8:overall simulation result which includes memory, data-processing, shift and some branch (b, bl , bi) instructions

Additional demonstration parts for the BEQ,BNE,BNC,BC

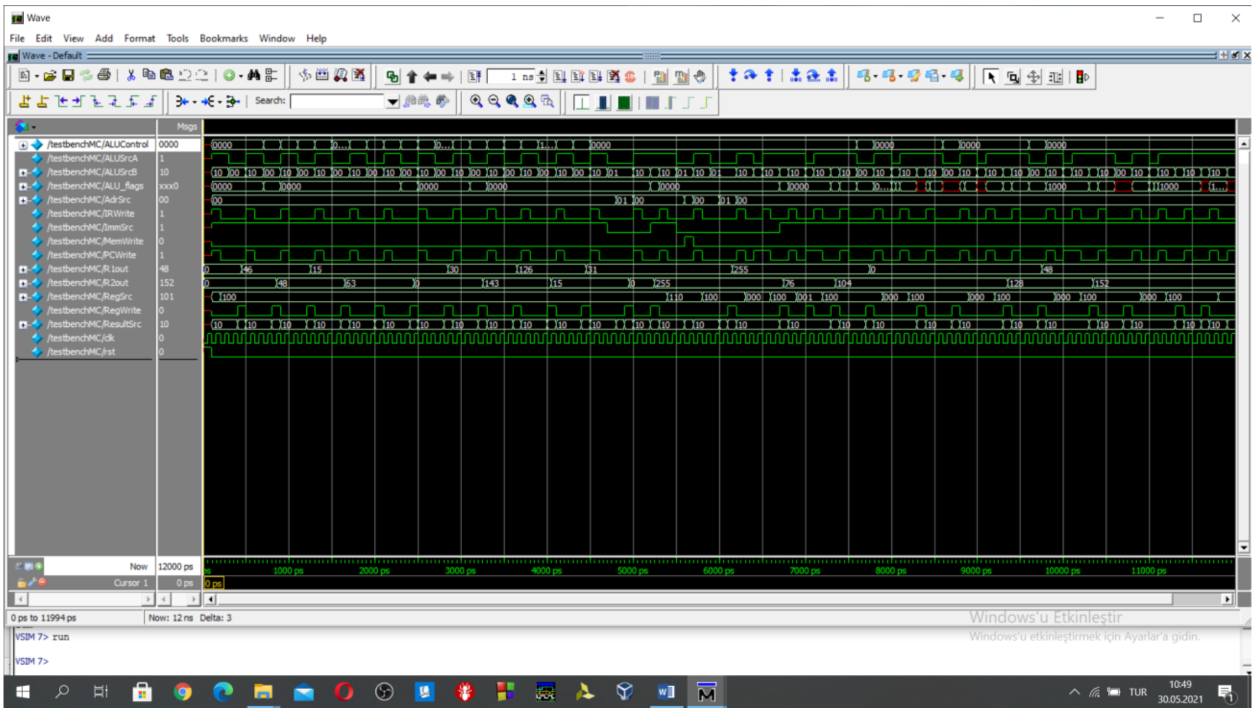


```
//B TO #76 branch here "B 76"
                    DATAmem[76] =  16'b1001_0010_0100_1100;//ldi r2 76
                    DATAmem[80] =  16'b1100_1000_0001_0000; //bl branch with link
to the 104
                    //BL TO THE 104 "BL 104"
                    DATAmem[104] =  16'b1001_0010_0110_1000;//ldi r2 104
                    DATAmem[108] =  16'b0001_0001_1011_0100;//sub  r1=r5-r5 0
                    DATAmem[112] =  16'b1101_1000_0000_1000; //beq then branch 128
                    //equal then branch here "BEQ 128" 128
                    DATAmem[128] =  16'b1001_0010_1000_0000;//ldi r2 128
                    DATAmem[132] =  16'b0001_0001_1010_1100;//sub  r1=r5-r3 not
equal

                    DATAmem[136] =  16'b1110_0000_0000_1000; //bne then branch 152

                    //not equal then "BNE 152" to 152
                    DATAmem[152]   =  16'b1001_0010_1001_1000;//ldi r2 152
                    DATAmem[156]   =  16'b0001_0001_1010_1100;//sub  r1=r5-r3 not
equal

                    DATAmem[160]   =  16'b1111_0000_0000_1000; //bnc


                    //branch "BNC 176" 176 if the carry flag is 0
                    DATAmem[176]   =  16'b1001_0010_1011_0000;//ldi r2 176
                    DATAmem[180]   =  16'b0000_0001_0100_1000;//add rd 1 rn 2 rm
2 initially r2=176

                    DATAmem[184]   =  16'b1110_1000_0000_1000;//bc if carry flag
is set

                    //bc control "BC 200"
                    DATAmem[200]   =  16'b1001_0010_1100_1000;//ldi r2 200

                    DATAmem[204]   =  16'b1001_0010_1101_1000;//ldi r2 216
```

After the discussion in the ODTUclass, some part of the instruction memory excluded. I add also BNC, BC, BEQ, BNE. I have controlled just the flags in testbench. I follow the similar steps while verifying previous testbench. Firstly, I use branch instruction, then I have demonstrated with LDI instructions.

Instruction at 108 activates the 0 flag. The beq branch works. The PC jumps to the address 128 in this address r2 is loaded with 128 as seen in Figure 9. R2 value is equal to 128 at time 9500 ps. That means BEQ works perfectly.

Instruction at 132 does not activate 0 flag. BNE works. The PC jumps to the address 152 in this address r2 is loaded with 152 as seen in Figure 9. R2 value is equal to 152 at time 10500 ps. That means BNE works perfectly.



```
/testbenchMC/ALUControl   0000
/testbenchMC/ALUSrcA      1
/testbenchMC/ALUSrcB      10
/testbenchMC/ALU_flags    xxx0
/testbenchMC/AdrSrc       00
/testbenchMC/IRWrite      1
/testbenchMC/ImmSrc       1
/testbenchMC/MemWrite     0
/testbenchMC/PCWrite      1
/testbenchMC/R1out        48
/testbenchMC/R2out        152
/testbenchMC/RegSrc       101
/testbenchMC/RegWrite     0
/testbenchMC/ResultSrc    10
/testbenchMC/clk          0
/testbenchMC/rst          0

Now       12000 ps
Cursor 1  0 ps

0 ps to 11994 ps    Now: 12 ns  Delta: 3

VSIM 7> run

VSIM 7>
```

Figure 9: BC, BNC, BEQ, BNE instructions added to the simulation

4. Datapath design Code and Testbench code