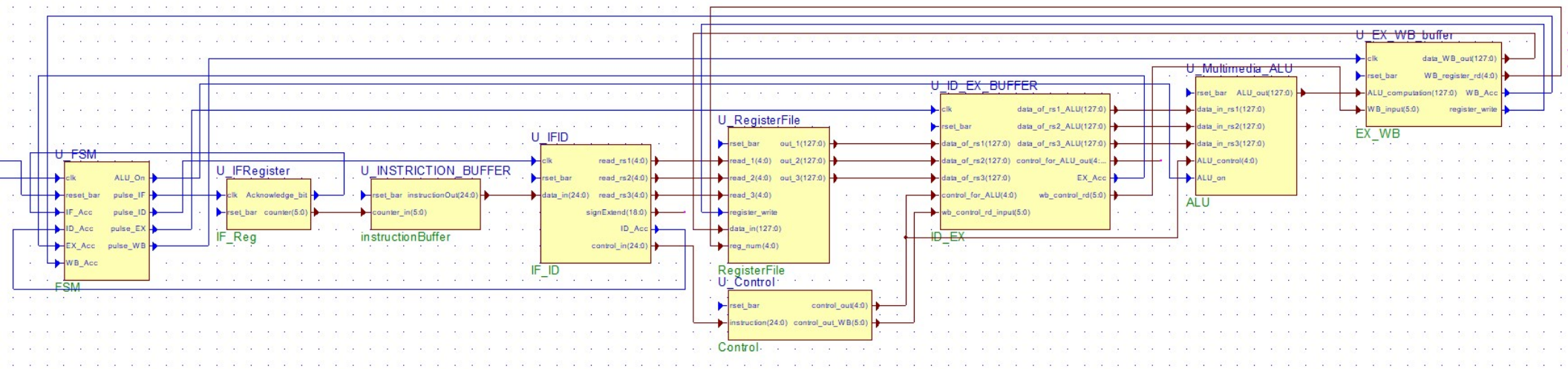
Asif Iqbal
Roni Das
ESE 345 Project
12/3/2019

The project is to design a four-stage pipelined multimedia unit. The multimedia instruction set was reduced. Several modules are needed for this design to operate as expected. The project was done in VHDL . Structural and behavioral coding style was used. Behavioral style contains multiple “process”. Each of these processes are are treated as a concurrent statement. All of the processes in an architecture are executed simultaneously. Structural style was used at the end to combine each of the modules to create a system.

For the whole system to work, individual modules were required, such as : Multimedia ALU, Register File, Instruction Buffer , Forwarding Unit, Four-Stage Pipelined Multimedia Unit(Clock edge-sensitive).In addition to these, our team decided to add a few additional components. Those are, a sign-extend, two mux and a control module. Sign extend module was used for the Load Immediate instruction.The two mux was used to decide between sign extend and index. The control was for the ALU unit. It supplied the ALU with control signals which dictated the computation ALU is to perform for a specific instruction.

The main goal of this project is to make the system work with forwarding. Each instruction is to be computed in four clock cycles. With forwarding, the number of clock cycles required to compute X instructions is significantly reduced. The system goes through four stages : IF, ID, EXE and WB. Without forwarding, following instruction has to wait for the current instruction to go through all four stages. With pipelining, when the current instruction goes to second stage, the following instruction goes to stage one. Given that two instructions are being processed, without forwarding, it will take ten clock cycles for both of the instruction. While on the other hand, with forwarding, it will only take five clock cycles.

Three instructions format was given by the instructor. The system has to work for these different types of instruction. Each format varies in terms of the instruction field. The first format was for Load Immediate. This is the only instruction in this format. The second format was R4 instruction format. In this format, computation such as Multiply-Add and Multiply-Subtract (signed/unsigned) is done. A total of eight instructions follows this format. Last but not least, R3-Instruction format. Multiplication, addition, subtraction, logical shifting/rotating took place. A total of twenty instructions followed this format.


U_FSM
clk
reset_bar
IF_Acc
ID_Acc
EX_Acc
WB_Acc
ALU_On
pulse_IF
pulse_ID
pulse_EX
pulse_WB
FSM
U_IFRegister
clk
reset_bar
Acknowledge_bit
counter(5:0)
IF_Reg
U_INSTRICTION_BUFFER
rset_bar
instructionOut(24:0)
counter_in(5:0)
instructionBuffer
U_IFID
clk
rset_bar
data_in(24:0)
read_rs1(4:0)
read_rs2(4:0)
read_rs3(4:0)
signExtend(18:0)
ID_Acc
control_in(24:0)
IF_ID
U_RegisterFile
rset_bar
read_1(4:0)
read_2(4:0)
read_3(4:0)
register_write
data_in(127:0)
reg_num(4:0)
out_1(127:0)
out_2(127:0)
out_3(127:0)
RegisterFile
U_Control
rset_bar
instruction(24:0)
control_out(4:0)
control_out_WB(5:0)
Control
U_ID_EX_BUFFER
clk
rset_bar
data_of_rs1(127:0)
data_of_rs2(127:0)
data_of_rs3(127:0)
control_for_ALU(4:0)
wb_control_rd_input(5:0)
data_of_rs1_ALU(127:0)
data_of_rs2_ALU(127:0)
data_of_rs3_ALU(127:0)
control_for_ALU_out(4:...
EX_Acc
wb_control_rd(5:0)
ID_EX
U_Multimedia_ALU
rset_bar
data_in_rs1(127:0)
data_in_rs2(127:0)
data_in_rs3(127:0)
ALU_control(4:0)
ALU_on
ALU_out(127:0)
ALU
U_EX_WB_buffer
clk
rset_bar
ALU_computation(127:0)
WB_input(5:0)
data_WB_out(127:0)
WB_register_rd(4:0)
WB_Acc
register_write
EX_WB

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : ALU
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30  entity ALU is
31      port(
32
33          rset_bar : in std_logic;                            --system Reset
34          data_in_rs1 : in std_logic_vector(127 downto 0);
35          data_in_rs2 : in std_logic_vector(127 downto 0);
36          data_in_rs3 : in std_logic_vector(127 downto 0);
37          ALU_control : in std_logic_vector(4 downto 0);
38
39
40          ALU_out : out  std_logic_vector(127 downto 0)
41      );
42  end ALU;
43
44
45
46  architecture behavioral of ALU is
47
48  type registers is array (64 downto 0) of std_logic_vector(24 downto 0);   --register
49
50  signal reg_array : registers;           -- singal is needed to access the array
51
52  signal ALU_calculation : std_logic_vector(127 downto 0);
53
54  signal add_sub : std_logic;
55
56  signal max_min : std_logic;
57
58  signal mul_mode : std_logic;
59
60  signal one_zero : std_logic;
61  signal rot_shift : std_logic_vector(1 downto 0);
62
63
64
65  function r4Format(rs3 : std_logic_vector ; rs2: std_logic_vector; rs1 : std_logic_vector;
    length: integer; mode: std_logic_vector) return std_logic_vector is
66
67      variable max : integer := (2**(length-1))-1;                                --
    32-bits-field max comparison
```

```
68      variable min : integer := (2**(length-1))*(-1);                                 --
    32-bits-field min comparison
69
70      -- variable  max_long : integer :=   9223372036854775807    ;
71      -- varoabe   min_long : integer := -9223372036854775808
72
73      variable result : integer ;                                                     -- result for
    Integer type
74      variable offset : integer := length/2 - 1;                                      -- offset to
    calculate arrya length
75
76      variable max_std :  std_logic_vector(63 downto 0):= x"7FFFFFFFFFFFFFFF";        --
    64-bits-field max comparison
77      variable min_std :  std_logic_vector(63 downto 0):= x"8000000000000000";        --
    64-bits-field min comparison
78      variable result_std:std_logic_vector(length-1 downto 0);                        -- result for
    Long type
79
80      begin
81
82
83          if(mode ="00") then
84              result := to_integer(signed(rs3(rs3'right+offset downto rs3'right)) * signed(rs2(
    rs2'right+offset downto rs2'right))) + to_integer(signed(rs1));    --ADD low
85              result_std :=  std_logic_vector( signed (rs3(rs3'right+offset downto rs3'right)) *
    signed( rs2(rs2'right+offset downto rs2'right)) + signed( rs1));
86
87          elsif(mode ="01") then
88              result := to_integer(signed(rs3(rs3'left downto rs3'left-offset)) * signed(rs2(rs2
    'left downto rs2'left-offset))) + to_integer(signed(rs1));              --ADD HIGH
89              result_std :=   std_logic_vector(signed(rs1) + signed(rs3(rs3'left downto rs3'left
    -offset)) * signed(rs2(rs2'left downto rs2'left-offset)));
90          elsif(mode ="10") then
91              result :=  to_integer(signed(rs1)) - to_integer(signed(rs3(rs3'right+offset downto
    rs3'right)) * signed(rs2(rs2'right+offset downto rs2'right)));      --SUB low
92              result_std :=  std_logic_vector(signed(rs1) - signed(rs3(rs3'right+offset downto
    rs3'right)) * signed(rs2(rs2'right+offset downto rs2'right)));
93          else
94              result := to_integer(signed(rs1)) - to_integer(signed(rs3(rs3'left downto rs3'left
    -offset)) * signed(rs2(rs2'left downto rs2'left-offset)));                  --SUB high
95              result_std := std_logic_vector(signed(rs1) - signed(rs3(rs3'left downto rs3'left-
    offset)) * signed(rs2(rs2'left downto rs2'left-offset)));
96           end if;
97
98
99          if( length = 32 and result <= min ) then
100             return std_logic_vector(to_signed(min,32));
101         elsif(length = 32 and result >= max)    then
102             return  std_logic_vector(to_signed(max,32));
103         elsif(length = 64 and signed(result_std) <= signed(min_std)) then
104             return min_std;
105         elsif(length = 64 and signed(result_std) >= signed(max_std)) then
106             return max_std;
107         else
108             return result_std;
109         end if;
110
111
112
113     end r4Format;
114
115 --AHS/SFHS
116 function saturate
117 (
118     half_word1 : std_logic_vector(15 downto 0);
119     half_word2 : std_logic_vector(15 downto 0);
120     add_sub : std_logic) return std_logic_vector is
121     variable output : std_logic_vector(15 downto 0);
122     variable added : std_logic_vector(15 downto 0);
123     variable subbed : std_logic_vector(15 downto 0);
124     variable upper_bound: integer := 32767;
```

```
125     variable lower_bound: integer := -32768;
126     begin
127         added := std_logic_vector(signed(half_word1) + signed(half_word2));
128         subbed := half_word2 - half_word1;
129         if(added < lower_bound) then
130             added := std_logic_vector(to_signed(lower_bound, 16));
131         elsif(added > upper_bound) then
132             added := std_logic_vector(to_signed(upper_bound, 16));
133         end if;
134 
135         if (subbed < lower_bound) then
136             subbed := std_logic_vector(to_signed(lower_bound, 16));
137         elsif (subbed > upper_bound) then
138             subbed := std_logic_vector(to_signed(upper_bound, 16));
139         end if;
140 
141 
142         if add_sub = '1' then
143             return added;
144         else return subbed;
145         end if;
146 end function;
147 
148 --function to count leading zeros
149 --CLZ
150 function CountZero
151 (
152     word : std_logic_vector(31 downto 0)) return std_logic_vector is
153     variable counter_out : std_logic_vector(31 downto 0);
154     variable count : integer := 0;
155     begin
156         for i in 31 downto 0 loop
157                 if word(i) = '0' then
158                     count := count + 1;
159                 else exit;
160                 end if;
161         end loop;
162         counter_out :=  std_logic_vector(to_unsigned(count, 32));
163         return counter_out;
164 end function;
165 
166 --POPCNTH
167 function CountOne
168 (
169     word : std_logic_vector(15 downto 0)) return std_logic_vector is
170     variable counter_out : std_logic_vector(15 downto 0);
171     variable count : integer := 0;
172     begin
173 
174         for i in 15 downto 0 loop
175             if word(i) = '1' then
176                 count := count + 1;
177             end if;
178         end loop;
179         counter_out :=  std_logic_vector(to_unsigned(count, 16));
180         return counter_out;
181 end function;
182 
183 
184 --MAX
185 function maxSignedWord
186 (
187     word1 : std_logic_vector(31 downto 0);
188     word2 : std_logic_vector(31 downto 0)) return std_logic_vector is
189     variable max : std_logic_vector(31 downto 0);
190 
191     begin
192         if(signed(word1) < signed(word2)) then
193             max := std_logic_vector( signed(word2));
194         elsif(signed(word1) > signed(word2)) then
195             max := std_logic_vector( signed(word1));
```

```
196         else max := std_logic_vector( signed(word1));
197         end if;
198         return max;
199 end function;
200
201 --min signed word
202 function minSignedWord
203 (
204     word1 : std_logic_vector(31 downto 0);
205     word2 : std_logic_vector(31 downto 0)) return std_logic_vector is
206     variable min : std_logic_vector(31 downto 0);
207
208     begin
209         if(signed(word1) < signed(word2)) then
210             min := std_logic_vector( signed(word1));
211         elsif(signed(word1) > signed(word2)) then
212             min := std_logic_vector( signed(word2));
213         else min := std_logic_vector( signed(word1));
214         end if;
215         return min;
216 end function;
217
218
219 --MSGN
220 function multiplySign
221 (
222     word1 : std_logic_vector(31 downto 0); word2 : std_logic_vector(31 downto 0)) return
    std_logic_vector is
223     variable mul_out : std_logic_vector(31 downto 0);
224     variable upper : integer := 2**31 - 1;
225     variable lower : integer := -2**31;
226
227 begin
228         if (to_integer(signed(word2)) < 0) then
229             mul_out := std_logic_vector(to_signed(  to_integer(signed(word1)) * (-1), 32));
230         else mul_out := word1;
231         end if;
232
233         if mul_out < lower then
234             mul_out := std_logic_vector(to_signed(lower, 32));
235         elsif mul_out > upper then
236             mul_out := std_logic_vector(to_signed(upper, 32));
237         end if;
238
239     return mul_out;
240 end function;
241
242 --MPYU
243 function multiplyUnsigned
244 (
245     word1 : std_logic_vector(31 downto 0); word2 : std_logic_vector(31 downto 0)
246     ) return std_logic_vector is
247     variable mul_out : std_logic_vector(31 downto 0);
248
249     begin
250         mul_out := std_logic_vector(word1(15 downto 0) * word2(15 downto 0));
251     return mul_out;
252 end function;
253
254
255 --function to rotate bits
256 --ROT/ROTW
257 function rotate
258 (
259     reg1 : std_logic_vector(127 downto 0);
260     reg2 : std_logic_vector(127 downto 0)) return std_logic_vector is
261     variable  result : std_logic_vector(127 downto 0);
262 begin
263         result := reg1;
264         for i in 0 to to_integer(unsigned(reg2(6 downto 0)))-1 loop
265             result := result(0) & result(127 downto 1);
```

```
266         end loop;
267         return result;
268 end function;
269 
270 function rotateWord
271 (
272     reg2 : std_logic_vector(127 downto 0)) return std_logic_vector is
273     variable  result : std_logic_vector(127 downto 0);
274 begin
275 
276         result := reg2;
277         for i in 0 to to_integer(unsigned(reg2(4 downto 0)))-1 loop
278             result(31 downto 0) := result(0) & result(31 downto 1);
279         end loop;
280         for i in 0 to to_integer(unsigned(reg2(35 downto 31)))-1 loop
281             result(63 downto 32) := result(32) & result(63 downto 33);
282         end loop;
283         for i in 0 to to_integer(unsigned(reg2(68 downto 64)))-1 loop
284             result(95 downto 64) := result(64) & result(95 downto 65);
285         end loop;
286         for i in 0 to to_integer(unsigned(reg2(99 downto 95)))-1 loop
287             result(127 downto 96) := result(96) & result(127 downto 97);
288         end loop;
289         return result;
290 end function;
291 
292 function shiftleft
293 (
294     reg1 : std_logic_vector(127 downto 0);
295     reg2 : std_logic_vector(127 downto 0)) return std_logic_vector is
296     variable  result : std_logic_vector(127 downto 0);
297 begin
298 
299         result := reg1;
300         for i in 0 to to_integer(unsigned(reg2(3 downto 0)))-1 loop
301             result(15 downto 0) := result(14 downto 0) & '0';
302         end loop;
303         for i in 0 to to_integer(unsigned(reg2(19 downto 16)))-1 loop
304             result(31 downto 16) := result(30 downto 16) & '0';
305         end loop;
306         for i in 0 to to_integer(unsigned(reg2(47 downto 44)))-1 loop
307             result(47 downto 32) := result(46 downto 32) & '0';
308         end loop;
309         for i in 0 to to_integer(unsigned(reg2(63 downto 60)))-1 loop
310             result(63 downto 48) := result(62 downto 48) & '0';
311         end loop;
312         for i in 0 to to_integer(unsigned(reg2(79 downto 76)))-1 loop
313             result(79 downto 64) := result(79 downto 65) & '0';
314         end loop;
315         for i in 0 to to_integer(unsigned(reg2(95 downto 92)))-1 loop
316             result(95 downto 80) := result(94 downto 80) & '0';
317         end loop;
318         for i in 0 to to_integer(unsigned(reg2(111 downto 108)))-1 loop
319             result(111 downto 96) := result(110 downto 96) & '0';
320         end loop;
321         for i in 0 to to_integer(unsigned(reg2(127 downto 124)))-1 loop
322             result(127 downto 112) := result(126 downto 112) & '0';
323         end loop;
324         return result;
325 end function;
326 
327 
328 begin
329 
330     p1: process(rset_bar,ALU_control,data_in_rs1,data_in_rs2,data_in_rs3)
331     --LI format---
332         variable index : integer := 0;
333         variable final_out : std_logic_vector(127 downto 0);
334         ----------------------
335     begin
336         if(rset_bar ='0') then
```

```
337             ALU_out <= (others => '0');
338         else
339
340             case ALU_control is
341
342                 when "00000" =>
343                         index := to_integer(unsigned(data_in_rs2));
344                         ALU_out (index*16+15 downto index*16) <= data_in_rs3(20 downto 5);
345                         final_out(index*16+15 downto index*16) :=  data_in_rs3(20 downto 5);

346
347
348             --------------
349             ----Format R3 : 4 words
350                 when "00001" =>
351
352                 for i in 0 to 3 loop
353               ALU_out(i*32+31 downto i*32) <= r4Format(data_in_rs3(i*32+31 downto i*32),
    data_in_rs2(i*32+31 downto i*32),data_in_rs1(i*32+31 downto i*32),32,"00" );
354
355                 end loop;
356
357                 when "00010" =>
358                 for i in 0 to 3 loop
359                 ALU_out(i*32+31 downto i*32) <= r4Format(data_in_rs3(i*32+31 downto i*32),
    data_in_rs2(i*32+31 downto i*32),data_in_rs1(i*32+31 downto i*32),32,"01" );
360
361                 end loop;
362
363                 when "00011" =>
364                 for i in 0 to 3 loop
365                 ALU_out(i*32+31 downto i*32) <= r4Format(data_in_rs3(i*32+31 downto i*32),
    data_in_rs2(i*32+31 downto i*32),data_in_rs1(i*32+31 downto i*32),32,"10" );
366
367                 end loop;
368
369                 when "00100" =>
370                 for i in 0 to 3 loop
371                 ALU_out(i*32+31 downto i*32) <= r4Format(data_in_rs3(i*32+31 downto i*32),
    data_in_rs2(i*32+31 downto i*32),data_in_rs1(i*32+31 downto i*32),32,"11" );

372                 end loop;
373             ---------------
374             -----Format R3 4 words Ends
375
376
377
378             --------------
379             ----Format R3 : 2 words
380                 when "00101" =>
381
382                 for i in 0 to 1 loop
383
384                  ALU_out(i*64+63 downto i*64) <= r4Format(data_in_rs3(i*64+63 downto i*64),
    data_in_rs2(i*64+63 downto i*64),data_in_rs1(i*64+63 downto i*64),64,"00" );
385
386                 end loop;
387
388                 when "00110" =>
389                 for i in 0 to 1 loop
390                ALU_out(i*64+63 downto i*64) <= r4Format(data_in_rs3(i*64+63 downto i*64),
    data_in_rs2(i*64+63 downto i*64),data_in_rs1(i*64+63 downto i*64),64,"01" );
391
392                 end loop;
393
394                 when "00111" =>
395                 for i in 0 to 1 loop
396                ALU_out(i*64+63 downto i*64) <= r4Format(data_in_rs3(i*64+63 downto i*64),
    data_in_rs2(i*64+63 downto i*64),data_in_rs1(i*64+63 downto i*64),64,"10" );
397
398                 end loop;
```

```
399
400                 when "01000" =>
401                 for i in 0 to 1 loop
402                 ALU_out(i*64+63 downto i*64) <= r4Format(data_in_rs3(i*64+63 downto i*64),
    data_in_rs2(i*64+63 downto i*64),data_in_rs1(i*64+63 downto i*64),64,"11" );

403                 end loop;
404             ---------------
405             -----Format R3 2 words Ends
406
407
408
409
410
411                 when "01001" =>  --NOP  TESTED/WORKS
412                 ALU_out <= (others =>'0');
413
414                 when "01010" =>   --A  TESTED/WORKS
415                 ALU_out(31 downto 0) <= data_in_rs1(31 downto 0) + data_in_rs2(31 downto 0);
416                 ALU_out(63 downto 32) <= data_in_rs1(63 downto 32) + data_in_rs2(63 downto 32
    );
417                 ALU_out(95 downto 64) <= data_in_rs1(95 downto 64) + data_in_rs2(95 downto 64
    );
418                 ALU_out(127 downto 96) <= data_in_rs1(127 downto 96) + data_in_rs2(127 downto
    96);
419
420                 when "01011" =>  --AH
421                 ALU_out(15 downto 0) <= data_in_rs1(15 downto 0) + data_in_rs2(15 downto 0);
422                 ALU_out(31 downto 16) <= data_in_rs1(31 downto 16) + data_in_rs2(31 downto 16
    );
423                 ALU_out(47 downto 32) <= data_in_rs1(47 downto 32) + data_in_rs2(47 downto 32
    );
424                 ALU_out(63 downto 48) <= data_in_rs1(63 downto 48) + data_in_rs2(63 downto 48
    );
425                 ALU_out(79 downto 64) <= data_in_rs1(79 downto 64) + data_in_rs2(79 downto 64
    );
426                 ALU_out(95 downto 80) <= data_in_rs1(95 downto 80) + data_in_rs2(95 downto 80
    );
427                 ALU_out(111 downto 96) <= data_in_rs1(111 downto 96) + data_in_rs2(111 downto
    96);
428                 ALU_out(127 downto 112) <= data_in_rs1(127 downto 112) + data_in_rs2(127
    downto 112);
429
430                 when "01100" => --AHS   //works
431                 add_sub <= '1'; --add
432                 ALU_out(15 downto 0) <= saturate( half_word1 => data_in_rs1(15 downto 0),
433                 half_word2 => data_in_rs2(15 downto 0), add_sub => add_sub);
434                 ALU_out(31 downto 16) <= saturate(half_word1 => data_in_rs1(31 downto 16),
435                 half_word2 => data_in_rs2(31 downto 16), add_sub => add_sub);
436                 ALU_out(47 downto 32) <= saturate(half_word1 => data_in_rs1(47 downto 32),
437                 half_word2 => data_in_rs2(47 downto 32), add_sub => add_sub);
438                 ALU_out(63 downto 48) <= saturate(half_word1 => data_in_rs1(63 downto 48),
439                 half_word2 => data_in_rs2(63 downto 48), add_sub => add_sub);
440
441                 ALU_out(79 downto 64) <= saturate(half_word1 => data_in_rs1(79 downto 64),
442                 half_word2 => data_in_rs2(79 downto 64), add_sub => add_sub);
443                 ALU_out(95 downto 80) <= saturate(half_word1 => data_in_rs1(95 downto 80),
444                 half_word2 => data_in_rs2(95 downto 80), add_sub => add_sub);
445                 ALU_out(111 downto 96) <= saturate(half_word1 => data_in_rs1(111 downto 96),
446                 half_word2 => data_in_rs2(111 downto 96), add_sub => add_sub);
447                 ALU_out(127 downto 112) <= saturate(half_word1 => data_in_rs1(127 downto 112),
448                 half_word2 => data_in_rs2(127 downto 112), add_sub => add_sub);
449
450
451                 when "01101" =>  --AND
452                 ALU_out <= (data_in_rs1 and data_in_rs2);
453
454                 when "01110" =>  --BCW  works
455                 ALU_out(31 downto 0) <= data_in_rs1(31 downto 0);
456                 ALU_out(63 downto 32) <= data_in_rs1(63 downto 32);
457                 ALU_out(95 downto 64) <= data_in_rs1(95 downto 64);
```

```
458                 ALU_out(127 downto 96) <= data_in_rs1(127 downto 96);
459
460                 when "01111" =>  --CLZ  //works
461                 one_zero <= '0';
462                 ALU_out(31 downto 0) <= CountZero(word => data_in_rs1(31 downto 0));
463                 ALU_out(63 downto 32) <= CountZero(word => data_in_rs1(63 downto 32));
464                 ALU_out(95 downto 64) <= CountZero(word => data_in_rs1(95 downto 64));
465                 ALU_out(127 downto 96) <= CountZero(word => data_in_rs1(127 downto 96));
466
467                 when "10000" =>  --MAX  //works
468                 ALU_out(31 downto 0) <= maxSignedWord(word1 => data_in_rs1(31 downto 0),
469                 word2 => data_in_rs2(31 downto 0));
470                 ALU_out(63 downto 32) <= maxSignedWord(word1 => data_in_rs1(63 downto 32),
471                 word2 => data_in_rs2(63 downto 32));
472                 ALU_out(95 downto 64) <= maxSignedWord(word1 => data_in_rs1(95 downto 64),
473                 word2 => data_in_rs2(95 downto 64));
474                 ALU_out(127 downto 96) <= maxSignedWord(word1 => data_in_rs1(127 downto 96),
475                 word2 => data_in_rs2(127 downto 96));
476
477
478                 when "10001" =>  --MIN  // works
479                 ALU_out(31 downto 0) <= minSignedWord(word1 => data_in_rs1(31 downto 0),
480                 word2 => data_in_rs2(31 downto 0));
481                 ALU_out(63 downto 32) <= minSignedWord(word1 => data_in_rs1(63 downto 32),
482                 word2 => data_in_rs2(63 downto 32));
483                 ALU_out(95 downto 64) <= minSignedWord(word1 => data_in_rs1(95 downto 64),
484                 word2 => data_in_rs2(95 downto 64));
485                 ALU_out(127 downto 96) <= minSignedWord(word1 => data_in_rs1(127 downto 96),
486                 word2 => data_in_rs2(127 downto 96));
487
488                 when "10010" =>  --MSGN
489                 mul_mode <= '1';
490                 ALU_out(31 downto 0) <= multiplySign(word1 => data_in_rs1(31 downto 0),
491                 word2 => data_in_rs2(31 downto 0));
492                 ALU_out(63 downto 32) <= multiplySign(word1 => data_in_rs1(63 downto 32),
493                 word2 => data_in_rs2(63 downto 32));
494                 ALU_out(95 downto 64) <= multiplySign(word1 => data_in_rs1(95 downto 64),
495                 word2 => data_in_rs2(95 downto 64));
496                 ALU_out(127 downto 96) <= multiplySign(word1 => data_in_rs1(127 downto 96),
497                 word2 => data_in_rs2(127 downto 96));
498
499                 when "10011" =>  --MPYU  works
500                 mul_mode <= '0';
501                 ALU_out(31 downto 0) <= multiplyUnsigned(word1 => data_in_rs1(31 downto 0),
502                 word2 => data_in_rs2(31 downto 0));
503                 ALU_out(63 downto 32) <= multiplyUnsigned(word1 => data_in_rs1(63 downto 32),
504                 word2 => data_in_rs2(63 downto 32));
505                 ALU_out(95 downto 64) <= multiplyUnsigned(word1 => data_in_rs1(95 downto 64),
506                 word2 => data_in_rs2(95 downto 64));
507                 ALU_out(127 downto 96) <= multiplyUnsigned(word1 => data_in_rs1(127 downto 96
    ),
508                 word2 => data_in_rs2(127 downto 96));
509
510
511
512                 when "10100" =>  --OR   works
513                 ALU_out <= (data_in_rs1 or data_in_rs2);
514
515
516                 when "10101" =>  --POPCNTH  works
517
518                 ALU_out(15 downto 0) <= CountOne(word => data_in_rs1(15 downto 0));
519                 ALU_out(31 downto 16) <= CountOne(word => data_in_rs1(31 downto 16));
520                 ALU_out(47 downto 32) <= CountOne(word => data_in_rs1(47 downto 32));
521                 ALU_out(63 downto 48) <= CountOne(word => data_in_rs1(63 downto 48));
522                 ALU_out(79 downto 64) <= CountOne(word => data_in_rs1(79 downto 64));
523                 ALU_out(95 downto 80) <= CountOne(word => data_in_rs1(95 downto 80));
524                 ALU_out(111 downto 96) <= CountOne(word => data_in_rs1(111 downto 96));
525                 ALU_out(127 downto 112) <= CountOne(word => data_in_rs1(127 downto 112));
526
527
```

```
528                 when "10110" =>  --ROT  //works
529                 ALU_out <= rotate(reg1 => data_in_rs1, reg2 => data_in_rs2);
530
531                 when "10111" =>  --ROTW  //works
532                 ALU_out <= rotateWord(reg2 => data_in_rs2);
533
534                 when "11000" =>  --SHLHI  //works
535                 ALU_out <= shiftLeft(reg1 => data_in_rs1, reg2 => data_in_rs2);
536
537                 when "11001" =>  --SFH
538                 ALU_out(15 downto 0) <= data_in_rs2(15 downto 0) - data_in_rs1(15 downto 0);
539                 ALU_out(31 downto 16) <= data_in_rs2(31 downto 16) - data_in_rs1(31 downto 16
    );
540                 ALU_out(47 downto 32) <= data_in_rs2(47 downto 32) - data_in_rs1(47 downto 32
    );
541                 ALU_out(63 downto 48) <= data_in_rs2(63 downto 48) - data_in_rs1(63 downto 48)
    ;
542                 ALU_out(79 downto 64) <= data_in_rs2(79 downto 64) - data_in_rs1(79 downto 64
    );
543                 ALU_out(95 downto 80) <= data_in_rs2(95 downto 80) - data_in_rs1(95 downto 80
    );
544                 ALU_out(111 downto 96) <= data_in_rs2(111 downto 96) - data_in_rs1(111 downto
    96);
545                 ALU_out(127 downto 112) <= data_in_rs2(127 downto 112) - data_in_rs1(127
    downto 112);
546                 --std_logic_vector(signed())
547                 when "11010" =>  --SFW
548                 ALU_out(31 downto 0) <= data_in_rs2(31 downto 0) - data_in_rs1(31 downto 0) ;
549                 ALU_out(63 downto 32) <= data_in_rs2(63 downto 32) - data_in_rs1(63 downto 32)
    ;
550                 ALU_out(95 downto 64) <= data_in_rs2(95 downto 64) - data_in_rs1(95 downto 64)
    ;
551                 ALU_out(127 downto 96) <= data_in_rs2(127 downto 96) - data_in_rs1(127 downto
    96) ;
552
553                 when "11011" =>  --SFHS
554                 add_sub <= '0';
555                 ALU_out(15 downto 0) <= saturate(half_word1 => data_in_rs1(15 downto 0),
556                 half_word2 => data_in_rs2(15 downto 0),add_sub => add_sub);
557                 ALU_out(31 downto 16) <= saturate(half_word1 => data_in_rs1(31 downto 16),
558                 half_word2 => data_in_rs2(31 downto 16),add_sub => add_sub);
559                 ALU_out(47 downto 32) <= saturate(half_word1 => data_in_rs1(47 downto 32),
560                 half_word2 => data_in_rs2(47 downto 32),add_sub => add_sub);
561                 ALU_out(63 downto 48) <= saturate(half_word1 => data_in_rs1(63 downto 48),
562                 half_word2 => data_in_rs2(63 downto 48),add_sub => add_sub);
563
564                 ALU_out(79 downto 64) <= saturate(half_word1 => data_in_rs1(79 downto 64),
565                 half_word2 => data_in_rs2(79 downto 64),add_sub => add_sub);
566                 ALU_out(95 downto 80) <= saturate(half_word1 => data_in_rs1(95 downto 80),
567                 half_word2 => data_in_rs2(95 downto 80),add_sub => add_sub);
568                 ALU_out(111 downto 96) <= saturate(half_word1 => data_in_rs1(111 downto 96),
569                 half_word2 => data_in_rs2(111 downto 96),add_sub => add_sub);
570                 ALU_out(127 downto 112) <= saturate(half_word1 => data_in_rs1(127 downto 112),
571                 half_word2 => data_in_rs2(127 downto 112),add_sub => add_sub);
572
573                 when "11100" =>  --XOR
574                 ALU_out <= data_in_rs1 xor data_in_rs2;
575
576
577                 when others => ALU_out <= (others => '0');
578             end case;
579            end if;
580     end process p1;
581 end behavioral;
582
583
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/Control.vhd

```
1    ----------------------------------------------------------------------------
2    --
3    -- Title       : MultimediaALU
4    -- Design      : SIMD
5    -- Author      : Asif
6    -- Company     : Stony Brook University
7    --
8    ----------------------------------------------------------------------------
9    --
10   -- File        : C:\Users\colle\OneDrive\SBU\Sixth
     Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\MultimediaALU.vhd
11   -- Generated   : Sat Nov 23 20:40:10 2019
12   -- From        : interface description file
13   -- By          : Itf2Vhdl ver. 1.22
14   --
15   ----------------------------------------------------------------------------
16   --
17   -- Description :
18   --
19   ----------------------------------------------------------------------------
20
21
22   library ieee;
23   use ieee.std_logic_1164.all;
24   use ieee.std_logic_unsigned.all;
25   use ieee.numeric_std.ALL;
26
27
28   entity Control is
29       port(
30       rset_bar : in std_logic;
31       instruction : in std_logic_vector(24 downto 0);
32
33
34       mux_select_1 : out std_logic;
35       mux_select_2 : out std_logic;
36       mux_select_Z : out std_logic;
37
38       control_out : out std_logic_vector(4 downto 0);
39       control_out_WB: out std_logic_vector(5 downto 0)
40
41       );
42   end Control;
43
44
45   architecture behave of Control is
46
47
48   begin
49       process(rset_bar,instruction)
50       begin
51           if (rset_bar ='0') then
52
53                       control_out <= (others => '0');
54                       control_out_WB <= (others => '0');
55                       mux_select_1 <= '0';
56                       mux_select_2 <= '0';
57                       mux_select_Z <= '0';
58
59           else
60
61                       control_out <= (others => '0');
62                       control_out_WB (5)<= '1';
63                       control_out_WB (4 downto 0) <= instruction(4 downto 0);
64                       mux_select_1 <= '0';
65                       mux_select_2 <= '0';
66                       mux_select_Z <= '0';
67
68                   if (instruction(24 downto 23) = "00" or
69                       instruction(24 downto 23) = "01")   then
70                       control_out <= std_logic_vector(to_unsigned(0,5)); --ldi
```

```
71                      mux_select_1 <= '1';
72                      mux_select_2 <= '1';
73
74                  elsif instruction(24 downto 23) = "10" then
75                      case instruction(22 downto 20) is
76                          when "000" => control_out <=    std_logic_vector(to_unsigned(1,5));
    --SIMALS
77                          when "001" => control_out <= std_logic_vector(to_unsigned(2,5));
    --SIMAHS
78                          when "010" => control_out <= std_logic_vector(to_unsigned(3,5));
    --SIMSLS
79                          when "011" => control_out <= std_logic_vector(to_unsigned(4,5));
    --SIMSHS
80                          when "100" => control_out <= std_logic_vector(to_unsigned(5,5));
    --SLIMALS
81                          when "101" => control_out <= std_logic_vector(to_unsigned(6,5));
    --SLIMAHS
82                          when "110" => control_out <= std_logic_vector(to_unsigned(7,5));
    --SLIMSLS
83                          when "111" => control_out <= std_logic_vector(to_unsigned(8,5));
    --SLIMSHS
84                          when others => control_out <= "ZZZZZ";
85                      end case;
86
87                  elsif instruction(24 downto 23) = "11" then
88                      case instruction(19 downto 15) is
89                          when "00000" => control_out <= std_logic_vector(to_unsigned(9,5));
    --NOP
90                                          control_out_WB (5)<= '0';
91                          when "00001" => control_out <= std_logic_vector(to_unsigned(10,5));
    --A
92                          when "00010" => control_out <= std_logic_vector(to_unsigned(11,5));
    --AH
93                          when "00011" => control_out <= std_logic_vector(to_unsigned(12,5));
    --AHS
94                          when "00100" => control_out<= std_logic_vector(to_unsigned(13,5));
    --AND
95                          when "00101" => control_out <= std_logic_vector(to_unsigned(14,5));
    --BCW
96                          when "00110" => control_out <= std_logic_vector(to_unsigned(15,5));
    --CLZ
97                          when "00111" => control_out <= std_logic_vector(to_unsigned(16,5));
    --MAX
98                          when "01000" => control_out <= std_logic_vector(to_unsigned(17,5));
    --MIN
99                          when "01001" => control_out <= std_logic_vector(to_unsigned(18,5));
    --MSGN
100                         when "01010" => control_out <= std_logic_vector(to_unsigned(19,5));
    --MPYU
101                         when "01011" => control_out <= std_logic_vector(to_unsigned(20,5));
    --OR
102                         when "01100" => control_out <= std_logic_vector(to_unsigned(21,5));
    --POPCNTN
103                         when "01101" => control_out <= std_logic_vector(to_unsigned(22,5));
    --ROT
104                         when "01110" => control_out <= std_logic_vector(to_unsigned(23,5));
    --ROTW
105                         when "01111" => control_out <= std_logic_vector(to_unsigned(24,5));
    --SHLHI
106                                         mux_select_Z<='1';
107                         when "10000" => control_out <= std_logic_vector(to_unsigned(25,5));
    --SFH
108                         when "10001" => control_out <= std_logic_vector(to_unsigned(26,5));
    --SFW
109                         when "10010" => control_out <= std_logic_vector(to_unsigned(27,5));
    --SFHS
110                         when "10011" => control_out <= std_logic_vector(to_unsigned(28,5));
    --XOR
111                         when others  => control_out <= "ZZZZZ";
112                     end case;
113                 end if;
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/Control.vhd

```
114               end if;
115       end process;
116
117   end behave;
118
```

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : ALU
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29  use STD.textio.all;
30  use ieee.std_logic_textio.all;
31
32
33  entity DATA_FW is
34
35      port(
36          rset_bar : in std_logic;                          --system Reset
37
38          register_num_1: in std_logic_vector(4 downto 0);
39          register_num_2: in std_logic_vector(4 downto 0);
40          register_num_3: in std_logic_vector(4 downto 0);
41
42          data_of_rs1 : in std_logic_vector(127 downto 0);
43          data_of_rs2 : in std_logic_vector(127 downto 0);
44          data_of_rs3 : in std_logic_vector(127 downto 0);
45
46          address_WB_of_rd : in std_logic_vector(5 downto 0);
47
48
49          ALU_calculation_for_FW : in std_logic_vector(127 downto 0);
50
51
52
53          fw_for_registers1 : out  std_logic_vector(127 downto 0);
54          fw_for_registers2 : out  std_logic_vector(127 downto 0);
55          fw_for_registers3 : out  std_logic_vector(127 downto 0)
56
57      );
58  end DATA_FW;
59
60
61
62  architecture behavioral of DATA_FW is
63
64
65
66  begin
67
68      P1: process(rset_bar, register_num_1, register_num_2, register_num_3, data_of_rs1,
    data_of_rs2,data_of_rs3, ALU_calculation_for_FW,address_WB_of_rd)
```

```
69
70
71      begin
72
73        if(rset_bar ='0') then
74
75              fw_for_registers1 <=(others =>'0');
76              fw_for_registers2 <=(others =>'0');
77              fw_for_registers3 <=(others =>'0');
78
79
80      elsif(address_WB_of_rd(5) = '0') then
81
82              fw_for_registers1 <= data_of_rs1;
83              fw_for_registers2 <= data_of_rs2;
84              fw_for_registers3 <= data_of_rs3;
85
86      else
87
88          if(  ( address_WB_of_rd(4 downto 0) = register_num_1 ) and ( address_WB_of_rd(4 downto
    0)= register_num_2 ) and( address_WB_of_rd(4 downto 0)= register_num_3 )) then
89
90              fw_for_registers1 <= ALU_calculation_for_FW;
91              fw_for_registers2 <= ALU_calculation_for_FW;
92              fw_for_registers3 <= ALU_calculation_for_FW;
93
94          elsif( (address_WB_of_rd(4 downto 0) = register_num_1) and (address_WB_of_rd(4 downto
    0)= register_num_2))  then
95              fw_for_registers1 <= ALU_calculation_for_FW;
96              fw_for_registers2 <= ALU_calculation_for_FW;
97              fw_for_registers3 <= data_of_rs3;
98
99          elsif(( address_WB_of_rd(4 downto 0) = register_num_1 )  and ( address_WB_of_rd(4
    downto 0)= register_num_3 ))  then
100             fw_for_registers1 <= ALU_calculation_for_FW;
101             fw_for_registers2 <= data_of_rs2;
102             fw_for_registers3 <=  ALU_calculation_for_FW;
103
104         elsif(( address_WB_of_rd(4 downto 0)= register_num_2 ) and( address_WB_of_rd(4 downto
    0)= register_num_3 )) then
105             fw_for_registers1 <= data_of_rs1;
106             fw_for_registers2 <= ALU_calculation_for_FW;
107             fw_for_registers3 <= ALU_calculation_for_FW;
108
109         elsif(address_WB_of_rd(4 downto 0) = (register_num_1))  then
110             fw_for_registers1 <= ALU_calculation_for_FW;
111             fw_for_registers2 <= data_of_rs2;
112             fw_for_registers3 <=  data_of_rs3;
113
114
115         elsif(address_WB_of_rd(4 downto 0) = (register_num_2))  then
116             fw_for_registers1 <= data_of_rs1;
117             fw_for_registers2 <= ALU_calculation_for_FW;
118             fw_for_registers3 <= data_of_rs3;
119
120         elsif(address_WB_of_rd(4 downto 0) = (register_num_3))  then
121             fw_for_registers1 <= data_of_rs1;
122             fw_for_registers2 <= data_of_rs2;
123             fw_for_registers3 <=  ALU_calculation_for_FW;
124         else
125             fw_for_registers1 <= data_of_rs1;
126             fw_for_registers2 <= data_of_rs2;
127             fw_for_registers3 <= data_of_rs3;
128            end if;
129
130
131         end if;
132
133
134     end process p1;
135
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/DataForward.vhd (/Top_Level/U_Forward)

```
136
137
138
139  end behavioral;
140
141
```

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : InstrictionDECODE/InstrutionEXecution
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30
31
32  entity ID_EX is
33
34      port(
35          clk : in std_logic;                                 -- system Clock
36          rset_bar : in std_logic;                            --system Reset
37
38
39          ---For Buffer---
40          data_of_rs1 : in std_logic_vector(127 downto 0);
41          data_of_rs2 : in std_logic_vector(127 downto 0);
42          data_of_rs3 : in std_logic_vector(127 downto 0);
43          control_for_ALU: in std_logic_vector(4 downto 0);
44          wb_control_rd_input : in std_logic_vector(5 downto 0);
45
46
47          index : in std_logic_vector(127 downto 0);
48          signExtend : in std_logic_vector(127 downto 0);
49
50          mux_control_1 : in std_logic;
51          mux_control_2 : in std_logic;
52          mux_control_Z : in std_logic;
53
54          rs2_c : in std_logic_vector(127 downto 0);
55
56
57          data_of_rs1_ALU : out std_logic_vector(127 downto 0);
58          data_of_rs2_ALU : out std_logic_vector(127 downto 0);
59          data_of_rs3_ALU : out std_logic_vector(127 downto 0);
60
61          --for control Unit for ALU-------
62          control_for_ALU_out: out std_logic_vector(4 downto 0);
63
64
65          index_out : out std_logic_vector(127 downto 0);
66          signExtend_out : out std_logic_vector(127 downto 0);
67
68          mux_control_1_out : out std_logic;
69          mux_control_2_out : out std_logic;
```

```
70          mux_control_Z_out : out std_logic;
71
72          rs2_out : out std_logic_vector(127 downto 0);
73
74          --for writeback control+ rd---
75          wb_control_rd : out std_logic_vector(5 downto 0)
76
77      );
78  end ID_EX;
79
80
81
82  architecture behavioral of ID_EX is
83
84      type registers is array (3 downto 0) of std_logic_vector(127 downto 0);    --register
85
86
87  signal reg_array : registers;           -- singal is needed to access the array
88
89
90  begin
91
92      process(clk,rset_bar)
93
94
95
96      variable webRegisters : std_logic_vector(5 downto 0);
97      variable indexVari : std_logic_vector(127 downto 0);
98      variable signExtendVari: std_logic_vector(127 downto 0);
99
100     variable mux_1_v : std_logic;
101     variable mux_2_v : std_logic;
102     variable mux_Z_v : std_logic;
103     variable rs2V : std_logic_vector(127 downto 0);
104
105     begin
106         if(rset_bar ='0') then
107             data_of_rs1_ALU <= (others =>'0');
108             data_of_rs2_ALU <= (others =>'0');
109             data_of_rs3_ALU <= (others =>'0');
110             control_for_ALU_out <= (others =>'0');
111             wb_control_rd <= (others =>'0');
112
113             mux_control_1_out <= '0';
114             mux_control_2_out <= '0';
115             mux_control_Z_out <= '0';
116
117             webRegisters := (others =>'0');
118             index_out <= (others =>'0');
119             signExtend_out <= (others =>'0');
120
121
122             signExtendVari := (others =>'0');
123             indexVari := (others =>'0');
124
125             rs2_out <= (others =>'0');
126             rs2V := (others =>'0');
127
128
129             mux_1_v :='0';
130             mux_2_v :='0';
131             mux_Z_v := '0';
132
133
134
135             for i in 0 to 3 loop
136                    reg_array(i) <=   (others =>'0');
137             end loop;
138
139
140
```

```
141
142
143         elsif (rising_edge(clk)) then
144
145
146
147             reg_array(0) <= data_of_rs1;
148             reg_array(1) <= data_of_rs2;
149             reg_array(2) <= data_of_rs3;
150             reg_array(3)(4 downto 0) <= control_for_ALU;
151             webRegisters := wb_control_rd_input;
152
153             indexVari :=  index;
154             signExtendVari := signExtend;
155
156             mux_1_v := mux_control_1;
157             mux_2_v := mux_control_2;
158
159             rs2V := rs2_c;
160             mux_Z_v := mux_control_Z;
161
162
163         end if;
164
165
166             data_of_rs1_ALU <= reg_array(0);
167             data_of_rs2_ALU <= reg_array(1);
168             data_of_rs3_ALU <= reg_array(2);
169             control_for_ALU_out <= reg_array(3)(4 downto 0);
170             wb_control_rd <= webRegisters;
171
172             index_out <= indexVari;
173             signExtend_out <= signExtendVari;
174
175             mux_control_1_out <= mux_1_v;
176             mux_control_2_out <= mux_2_v;
177             mux_control_Z_out <= mux_Z_v;
178             rs2_out <= rs2V;
179
180
181
182       end process;
183
184
185
186
187
188 end behavioral;
189
190
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/ID_IERegister_TB.vhd

```
1   ----------------------------------------------------------------------------
2   --
3   -- Title       : ID_EX_TB
4   -- Design      : SIMD
5   -- Author      : Asif
6   -- Company     : Stony Brook University
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\Asif\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined SIMD Multimedia Unit\src\RegisterFile TB.vhd
11  -- Generated   : Sat Nov 23 06:47:06 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile_TB} architecture {RegisterFile_TB}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30
31
32  entity ID_EX_TB is
33
34  end ID_EX_TB;
35
36  --}} End of automatically maintained section
37
38  architecture behavioral_TB of ID_EX_TB is
39
40
41  --input signals
42  signal clk, rset_bar: std_logic;
43
44
45          ---For Buffer---
46  signal      data_of_rs1 : std_logic_vector(127 downto 0);
47  signal      data_of_rs2 :  std_logic_vector(127 downto 0);
48  signal      data_of_rs3 : std_logic_vector(127 downto 0);
49  signal      control_for_ALU: std_logic_vector(4 downto 0);
50
51  signal      data_of_rs1_ALU :  std_logic_vector(127 downto 0);
52  signal      data_of_rs2_ALU :  std_logic_vector(127 downto 0);
53  signal      data_of_rs3_ALU : std_logic_vector(127 downto 0);
54
55          --for control Unit for ALU-------
56  signal      control_for_ALU_out: std_logic_vector(4 downto 0);
57
58
59
60
61
62
63
64  signal temp : unsigned(24 downto 0);
65
66  constant period : time :=10 ns;
67
68
69  begin
70      U_ID_EX_BUFFER: entity ID_IE
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/ID_IERegister_TB.vhd

```
71          port map(clk => clk, rset_bar => rset_bar, data_of_rs1 => data_of_rs1,data_of_rs2 =>
    data_of_rs2,data_of_rs3 => data_of_rs3, control_for_ALU => control_for_ALU,
72                  data_of_rs1_ALU => data_of_rs1_ALU, data_of_rs2_ALU => data_of_rs2_ALU,
    data_of_rs3_ALU => data_of_rs3_ALU, control_for_ALU_out => control_for_ALU_out);
73
74
75
76      process
77
78      begin
79
80          rset_bar <= '0', '1' after 5ns;
81
82          clk <='1';
83
84
85          for i in 0 to 64 loop
86
87              --temp <= to_unsigned(i**2,25);
88      data_of_rs1 <= std_logic_vector(to_unsigned((i*9+7),128));
89      data_of_rs2 <= std_logic_vector(to_unsigned(i*2,128));
90      data_of_rs3 <= std_logic_vector(to_unsigned(i*7+3,128));
91      control_for_ALU <= std_logic_vector(to_unsigned(i,5));
92
93
94              wait for period/2;
95              --data_in <= std_logic_vector(temp);
96
97
98              clk <= not clk;
99          end loop;
100
101
102     wait;
103     end process;
104
105
106
107
108 end behavioral_TB;
109
```

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : RegisterFile
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29  use STD.textio.all;
30  use ieee.std_logic_textio.all;
31
32
33  entity instructionBuffer is
34
35      port(
36
37          rset_bar : in std_logic;                          --system Reset
38
39          counter_in : in std_logic_vector(5 downto 0);
40          instructionOut : out  std_logic_vector(24 downto 0)
41      );
42  end instructionBuffer;
43
44
45
46  architecture behavioral of instructionBuffer is
47
48  type registers is array (64 downto 0) of std_logic_vector(24 downto 0);   --register
49
50  signal reg_array : registers;           -- singal is needed to access the array
51
52
53  begin
54
55      process(rset_bar,counter_in)
56
57      ---For File Input SETUP----
58      File f: text;
59      constant filename : string :="instructionsOutput.txt";
60      Variable L : Line;
61      variable i : integer :=0;
62      variable b: std_logic_vector(24 downto 0);
63      ------------------------------
64
65
66      begin
67          if(rset_bar ='0') then
68              instructionOut <= (others =>'0');
69
```

```
70              --File read----
71
72            file_open(f,filename,read_mode);
73            while((i<=63) and (not endfile(f))) loop
74                readline(f,l);
75                read(l,b);
76                reg_array(i) <= b;
77                i := i+1;
78            end loop;
79
80
81
82            while(i<=63) loop
83                reg_array(i) <= "11000000000000000000000000";
84                i := i+1;
85            end loop;
86            file_close(f);
87
88                ----------------
89            end if;
90
91            instructionOut <= reg_array(to_integer(unsigned(counter_in)));
92
93          end process;
94
95
96
97
98
99    end behavioral;
100
101
```

```
1   ----------------------------------------------------------------------------
2   --
3   -- Title       : RegisterFile_TB
4   -- Design      : SIMD
5   -- Author      : Asif
6   -- Company     : Stony Brook University
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\Asif\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined SIMD Multimedia Unit\src\RegisterFile TB.vhd
11  -- Generated   : Sat Nov 23 06:47:06 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile_TB} architecture {RegisterFile_TB}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29  use STD.textio.all;
30  use ieee.std_logic_textio.all;
31
32
33  entity InstructionBuffer_TB is
34
35  end InstructionBuffer_TB;
36
37  --}} End of automatically maintained section
38
39  architecture behavioral_TB of InstructionBuffer_TB is
40
41  type mem is array (28 downto 0) of std_logic_vector(24 downto 0);
42  signal t_mem: mem;
43
44
45  signal clk, rset_bar: std_logic;
46  --signal data_in : std_logic_vector(24 downto 0);
47  signal  instructionOut : std_logic_vector(24 downto 0);
48
49
50  begin
51      UInsBuff: entity instructionBuffer
52                  port map(counter_in => counter_in, rset_bar => rset_bar, instructionOut =>
    instructionOut);
53
54
55      process
56
57      File f: text;
58      constant filename : string :="instructionsOutput.txt";
59      Variable L : Line;
60      variable i : integer :=0;
61      variable b: std_logic_vector(24 downto 0);
62      begin
63
64          rset_bar <= '0', '1' after 5ns;
65
66          file_open(f,filename,read_mode);
67          while((i<=63) and (not endfile(f))) loop
68              readline(f,l);
69              read(l,b);
```

```
70              t_mem(i) <= b;
71              i := i+1;
72          end loop;
73          file_close(f);
74
75          clk <='1';
76
77          for i in 0 to 64 loop
78              wait for 10ns;
79              clk <= not clk;
80          end loop;
81
82
83      wait;
84      end process;
85
86
87
88
89  end behavioral_TB;
90
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/IF_Register.vhd

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : InstrictionFetch/InstrutionDecoder
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30
31
32  entity IF_Reg is
33
34      port(
35          clk : in std_logic;                                 -- system Clock
36          rset_bar : in std_logic;                            --system Reset
37
38          counter : out std_logic_vector( 5 downto 0)
39
40      );
41  end IF_Reg;
42
43
44
45  architecture behavioral of IF_Reg is
46
47
48  begin
49
50      process(clk,rset_bar)
51
52
53      variable count : integer range 0 to 2**6-1;
54
55      begin
56          if(rset_bar ='0') then
57              count := 0;
58              counter <= (others =>'0');
59
60          elsif (rising_edge(clk)) then
61              if( not (count = 63) ) then
62                 count := count +1;
63
64              end if;
65
66          end if;
67
68
69          counter <= std_logic_vector(to_unsigned(count,6));
```

```
70        end process;
71
72
73
74
75
76    end behavioral;
77
78
```

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : InstrictionFetch/InstrutionDecoder
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30
31
32  entity IF_ID is
33
34      port(
35          clk : in std_logic;                                 -- system Clock
36          rset_bar : in std_logic;                            --system Reset
37          data_in : in std_logic_vector(24 downto 0);
38
39          ---For registerFile---
40          read_rs1 : out std_logic_vector(4 downto 0);
41          read_rs2 : out std_logic_vector(4 downto 0);
42          read_rs3 : out std_logic_vector(4 downto 0);
43
44
45          ---for Signed extend + indexLoading
46          signExtend: out std_logic_vector(18 downto 0);
47
48
49          --for control Unit-------
50          control_in : out std_logic_vector(24 downto 0)
51
52      );
53  end IF_ID;
54
55
56
57  architecture behavioral of IF_ID is
58
59
60
61  begin
62
63      process(clk,rset_bar)
64
65      variable reg_array : std_logic_vector(24 downto 0);
66
67
68      begin
69          if(rset_bar ='0') then
```

```
70              read_rs1 <= (others =>'0');
71              read_rs2 <= (others =>'0');
72              read_rs3 <= (others =>'0');
73              signExtend <= (others =>'0');
74              reg_array := (others =>'0');
75              control_in <= (others => '0');
76
77          elsif (rising_edge(clk)) then
78
79
80              reg_array := data_in;
81
82
83          end if;
84
85              read_rs1 <= reg_array(9 downto 5);
86              read_rs2 <= reg_array(14 downto 10);
87              read_rs3 <= reg_array(19 downto 15);
88              signExtend <= reg_array(23 downto 5);
89              control_in <= reg_array;
90        end process;
91
92
93
94
95
96  end behavioral;
97
98
```

```
1
    ------------------------------------------------------------------------
2   --
3   -- Title       : D-ff
4   -- Design      : lab07
5   --
6   -- Author      : Asif Iqbal => ID: 110333685
7   --             : Roni Das   => ID: 108378223
8   --
9   -- Company     : Stony Brook
10  --
11  ------------------------------------------------------------------------
12  --
13  -- File        : C:\Users\Asif\OneDrive\SBU\Fifth Semester\ESE382\Labs\LAB07\LAB07\src\D-ff.vhd
14  -- Generated   : Tue Mar  26 14:57:59 2019
15  -- From        : interface description file
16  -- By          : Itf2Vhdl ver. 1.22
17  --
18  ------------------------------------------------------------------------
19  --
20  -- Description :The code below is for a D-Flip-Flop memory element. Using
21  --              Behavioral style coding dff is described. During every
22  --              rising clock edge qff clocks in values on data line d.
23  --
24  -- Input: d, clk; Type: STD_LOGIC;
25  -- Output: qff; Type: STD_LOGIC;
26
27  -- Lab07: Task1         Bench:02        Section: 02
28  ------------------------------------------------------------------------
29
30  --{{ Section below this comment is automatically maintained
31  --   and may be overwritten
32  --{entity {therm2gray} architecture {therm2grayIf}}
33
34
35  library IEEE;
36  use IEEE.std_logic_1164.all;
37  use ieee.numeric_std.all;
38
39
40  entity mux_2to1 is
41      port(
42          rset_bar : in std_logic;
43          data_rs : in std_logic_vector(127 downto 0); -- data input channels
44          data_buff: in std_logic_vector(127 downto 0); -- data input channels
45
46          cs : in std_logic; -- mux select inputs
47          mux_out : out std_logic_vector(127 downto 0) -- output of mux
48          );
49  end mux_2to1;
50
51
52  architecture behavioral of mux_2to1 is
53      begin
54
55          process (rset_bar,cs,data_rs,data_buff)
56          begin
57              if(rset_bar ='0') then
58                  mux_out <= (others =>'0');
59
60              else
61
62              case cs is
63                  when '1'  => mux_out <= data_buff;
64                  when '0'  => mux_out <= data_rs;
65                  when others => mux_out <= data_rs;
66
67              end case;
68              end if;
69           end process;
70
```

```
71
72
73  end behavioral;
74
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/RegisterFile.vhd (/Top_Level/U_RegisterFile)

```
1    ----------------------------------------------------------------------------
2    --
3    -- Title       : RegisterFile
4    -- Design      : SIMD
5    -- Author      :
6    -- Company     :
7    --
8    ----------------------------------------------------------------------------
9    --
10   -- File        : C:\Users\colle\OneDrive\SBU\Sixth
     Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11   -- Generated   : Sat Nov 23 03:16:03 2019
12   -- From        : interface description file
13   -- By          : Itf2Vhdl ver. 1.22
14   --
15   ----------------------------------------------------------------------------
16   --
17   -- Description :
18   --
19   ----------------------------------------------------------------------------
20
21
22   library ieee;
23   use ieee.std_logic_1164.all;
24   use ieee.std_logic_unsigned.all;
25   use ieee.numeric_std.ALL;
26
27
28   entity RegisterFile is
29       port(
30           rset_bar : in std_logic;
31
32           read_1 : in std_logic_vector(4 downto 0);        --targetted registers
33           read_2 : in std_logic_vector(4 downto 0);        --to read from and write to
34           read_3 : in std_logic_vector(4 downto 0);
35
36           register_write : in std_logic;                   --control for write
37           data_in : in std_logic_vector(127 downto 0);     --data to be written
38           reg_num : in std_logic_vector(4 downto 0);       --the register number
39
40
41           out_1 : out  std_logic_vector(127 downto 0);
42           out_2 : out  std_logic_vector(127 downto 0);
43           out_3 : out  std_logic_vector(127 downto 0)
44
45
46
47       );
48   end RegisterFile;
49
50
51
52   architecture behavioral of RegisterFile is
53
54   type registers is array (31 downto 0) of std_logic_vector(127 downto 0);   --register
55
56   signal reg_array : registers;                                               -- singal is
     needed to access the array
57
58   begin
59       process(rset_bar, register_write,read_1,read_2,read_3,reg_num,data_in )
60       begin
61           if (rset_bar = '0') then
62
63               for i in 0 to 31 loop
64                   reg_array(i) <=     std_logic_vector(to_unsigned(i*2,128));
65               end loop;
66               out_1 <= (others => '0');
67               out_2 <= (others => '0');
68               out_3 <= (others => '0');
69           elsif( register_write = '1' ) then
```

```
70
71              reg_array(to_integer(unsigned(reg_num))) <= data_in;
72
73    -----
74
75          if(  ( reg_num = read_1 ) and ( reg_num = read_2 ) and( reg_num = read_3 )) then
76
77              out_1 <= data_in;
78              out_2 <= data_in;
79              out_3 <= data_in;
80
81          elsif( ( reg_num = read_1 ) and ( reg_num = read_2 ))   then
82
83              out_1 <= data_in;
84              out_2 <= data_in;
85              out_3 <= reg_array(to_integer(unsigned(read_3)));
86
87          elsif(( reg_num = read_1 ) and( reg_num = read_3 )) then
88
89              out_1 <= data_in;
90              out_2 <= reg_array(to_integer(unsigned(read_2)));
91              out_3 <= data_in;
92
93          elsif(( reg_num = read_2 ) and( reg_num = read_3 )) then
94
95              out_1 <= reg_array(to_integer(unsigned(read_1)));
96              out_2 <= data_in;
97              out_3 <= data_in;
98
99
100         elsif(( reg_num = read_1 )) then
101
102             out_1 <= data_in;
103             out_2 <= reg_array(to_integer(unsigned(read_2)));
104             out_3 <= reg_array(to_integer(unsigned(read_3)));
105
106
107         elsif(( reg_num = read_2 )) then
108
109             out_1 <= reg_array(to_integer(unsigned(read_1)));
110             out_2 <= data_in;
111             out_3 <= reg_array(to_integer(unsigned(read_3)));
112
113         elsif(( reg_num = read_3 )) then
114
115             out_1 <= reg_array(to_integer(unsigned(read_1)));
116             out_2 <= reg_array(to_integer(unsigned(read_2)));
117             out_3 <= data_in;
118         else
119
120             out_1 <= reg_array(to_integer(unsigned(read_1)));
121             out_2 <= reg_array(to_integer(unsigned(read_2)));
122             out_3 <= reg_array(to_integer(unsigned(read_3)));
123            end if;
124
125   -----
126
127         else
128             out_1 <= reg_array(to_integer(unsigned(read_1)));
129             out_2 <= reg_array(to_integer(unsigned(read_2)));
130             out_3 <= reg_array(to_integer(unsigned(read_3)));
131
132             end if;
133     end process;
134
135
136
137
138   end behavioral;
139
140
```

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : Signed Extend and Index Component
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30
31
32  entity signIndex is
33
34      port(
35          rset_bar : in std_logic;                            --system Reset
36          data_in_ext :  in std_logic_vector(18 downto 0);
37
38          index : out  std_logic_vector(127 downto 0);
39          rs2 : out std_logic_vector(127 downto 0);
40          signExtendOutput: out std_logic_vector(127 downto 0)
41      );
42  end signIndex;
43
44
45
46  architecture behavioral of signIndex is
47
48
49  begin
50
51      process(rset_bar, data_in_ext)
52
53
54      begin
55          if(rset_bar ='0') then
56              index <= (others =>'0');
57              signExtendOutput <= (others =>'0');
58              rs2 <= (others =>'0');
59
60          else
61
62              index(2 downto 0) <= data_in_ext(18 downto 16);
63              signExtendOutput(20 downto 5) <= data_in_ext(15 downto 0);
64              rs2(4 downto 0) <= data_in_ext(9 downto 5);
65              end if;
66        end process;
67
68
69
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/SignExtend_Index.vhd (/Top_Level/U_SignExtend_In

```
70
71
72  end behavioral;
73
74
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/EX_WBRegister.vhd

```
1
    ----------------------------------------------------------------------------
2   --
3   -- Title       : InstrictionExecute/WriteBack
4   -- Design      : SIMD
5   -- Author      :
6   -- Company     :
7   --
8   ----------------------------------------------------------------------------
9   --
10  -- File        : C:\Users\colle\OneDrive\SBU\Sixth
    Semester\ESE345\Project\VHDL\Pipelined_SIMD_Multimedia_Unit\src\RegisterFile.vhd
11  -- Generated   : Sat Nov 23 03:16:03 2019
12  -- From        : interface description file
13  -- By          : Itf2Vhdl ver. 1.22
14  --
15  ----------------------------------------------------------------------------
16  --
17  -- Description :
18  --
19  ----------------------------------------------------------------------------
20
21  --{{ Section below this comment is automatically maintained
22  --   and may be overwritten
23  --{entity {RegisterFile} architecture {behavioral}}
24
25  library ieee;
26  use ieee.std_logic_1164.all;
27  use ieee.std_logic_unsigned.all;
28  use ieee.numeric_std.ALL;
29
30
31
32  entity EX_WB is
33
34      port(
35          clk : in std_logic;                                  -- system Clock
36          rset_bar : in std_logic;                             --system Reset
37
38
39          ---For registerFile---
40          ALU_computation : in std_logic_vector(127 downto 0);
41          WB_input : in std_logic_vector(5 downto 0);
42
43
44          ----WriteBackData
45          data_WB_out: out std_logic_vector(127 downto 0);
46          WB_register_rd: out std_logic_vector(4 downto 0);
47
48
49          --writeBackControl---
50          register_write : out std_logic
51
52      );
53  end EX_WB;
54
55
56
57  architecture behavioral of EX_WB is
58
59
60  begin
61
62      process(clk,rset_bar)
63
64      variable DATA_reg : std_logic_vector(127 downto 0);
65      variable control_reg : std_logic;
66      variable rd_reg : std_logic_vector(4 downto 0);
67
68      begin
69          if(rset_bar ='0') then
```

```
70              data_WB_out <= (others =>'0');
71              WB_register_rd <= (others =>'0');
72              register_write <= '0';
73
74
75                  control_reg := '0';
76                  DATA_reg :=  (others =>'0');
77                  rd_reg := (others => '0');
78
79
80
81
82          elsif (rising_edge(clk)) then
83
84
85
86              control_reg := WB_input(5);
87              DATA_reg :=  ALU_computation;
88              rd_reg := WB_input(4 downto 0);
89
90
91          end if;
92              register_write <= control_reg;
93              WB_register_rd <= rd_reg;
94              data_WB_out <= DATA_reg;
95
96
97        end process;
98
99
100
101
102
103 end behavioral;
104
105
```

```
1    ----------------------------------------------------------------------------
2    --
3    -- Title       : EX_WB_TB
4    -- Design      : SIMD
5    -- Author      : Asif
6    -- Company     : Stony Brook University
7    --
8    ----------------------------------------------------------------------------
9    --
10   -- File        : C:\Users\Asif\OneDrive\SBU\Sixth
     Semester\ESE345\Project\VHDL\Pipelined SIMD Multimedia Unit\src\RegisterFile TB.vhd
11   -- Generated   : Sat Nov 23 06:47:06 2019
12   -- From        : interface description file
13   -- By          : Itf2Vhdl ver. 1.22
14   --
15   ----------------------------------------------------------------------------
16   --
17   -- Description :
18   --
19   ----------------------------------------------------------------------------
20
21   --{{ Section below this comment is automatically maintained
22   --   and may be overwritten
23   --{entity {RegisterFile_TB} architecture {RegisterFile_TB}}
24
25   library ieee;
26   use ieee.std_logic_1164.all;
27   use ieee.std_logic_unsigned.all;
28   use ieee.numeric_std.ALL;
29
30
31
32   entity EX_WB_TB is
33
34   end EX_WB_TB;
35
36   --}} End of automatically maintained section
37
38   architecture behavioral_TB of EX_WB_TB is
39
40
41   --input signals
42   signal clk, rset_bar: std_logic;
43
44
45           ---For registerFile---
46   signal      ALU_computation : std_logic_vector(127 downto 0);
47   signal      WB_input : std_logic_vector(5 downto 0);
48
49
50           ----WriteBackData
51   signal      data_WB_out: std_logic_vector(127 downto 0);
52   signal      WB_register_rd: std_logic_vector(4 downto 0);
53
54           --writeBackControl---
55   signal      register_write : std_logic;
56
57
58
59
60
61
62   signal temp : unsigned(24 downto 0);
63
64   constant period : time :=10 ns;
65
66
67   begin
68       U_EX_WB_BUFFER: entity EX_WB
69           port map(clk => clk, rset_bar => rset_bar, ALU_computation => ALU_computation,
     WB_input => WB_input, data_WB_out => data_WB_out,
```

```
                    WB_register_rd => WB_register_rd , register_write => register_write);



    process

    begin

        rset_bar <= '0', '1' after 5ns;

        clk <='1';


        for i in 0 to 64 loop

            --temp <= to_unsigned(i**2,25);
    ALU_computation <= std_logic_vector(to_unsigned((i*9+7),128));
    WB_input <= std_logic_vector(to_unsigned(i*2,6));


            wait for period/2;
            --data_in <= std_logic_vector(temp);


            clk <= not clk;
        end loop;


    wait;
    end process;



end behavioral_TB;

```

```
1    ----------------------------------------------------------------------------
2    --
3    -- Title       : IF_ID_TB
4    -- Design      : SIMD
5    -- Author      : Asif
6    -- Company     : Stony Brook University
7    --
8    ----------------------------------------------------------------------------
9    --
10   -- File        : C:\Users\Asif\OneDrive\SBU\Sixth
     Semester\ESE345\Project\VHDL\Pipelined SIMD Multimedia Unit\src\RegisterFile TB.vhd
11   -- Generated   : Sat Nov 23 06:47:06 2019
12   -- From        : interface description file
13   -- By          : Itf2Vhdl ver. 1.22
14   --
15   ----------------------------------------------------------------------------
16   --
17   -- Description :
18   --
19   ----------------------------------------------------------------------------
20
21   --{{ Section below this comment is automatically maintained
22   --   and may be overwritten
23   --{entity {RegisterFile_TB} architecture {RegisterFile_TB}}
24
25   library ieee;
26   use ieee.std_logic_1164.all;
27   use ieee.std_logic_unsigned.all;
28   use ieee.numeric_std.ALL;
29   use STD.textio.all;
30   use ieee.std_logic_textio.all;
31
32
33
34
35   entity Top_Level is
36
37   end Top_Level;
38
39   --}} End of automatically maintained section
40
41   architecture behavioral_TB of Top_Level is
42
43
44   --------Functions to write one TEXT file--------------------
45
46       function chr(sl: std_logic) return character is
47           variable c: character;
48           begin
49           case sl is
50           when 'U' => c:= 'U';
51           when 'X' => c:= 'X';
52           when '0' => c:= '0';
53           when '1' => c:= '1';
54           when 'Z' => c:= 'Z';
55           when 'W' => c:= 'W';
56           when 'L' => c:= 'L';
57           when 'H' => c:= 'H';
58           when '-' => c:= '-';
59           end case;
60       return c;
61       end chr;
62
63
64       function str(slv: std_logic_vector) return string is
65           variable result : string (1 to slv'length);
66           variable r : integer;
67           begin
68               r := 1;
69               for i in slv'range loop
70               result(r) := chr(slv(i));
```

```
71              r := r + 1;
72              end loop;
73          return result;
74      end str;
75
76   ------------------------------------------------------------
77
78
79
80   --input signals  to System
81   signal clk, rset_bar: std_logic;
82
83
84   --For Instruction Buffer---
85   signal      data_in : std_logic_vector(24 downto 0);
86   signal      data_in_ext : std_logic_vector(18 downto 0);
87
88
89
90
91          --For RegisterFile
92
93   signal      read_1 :  std_logic_vector(4 downto 0);      --targetted registers
94   signal      read_2 :  std_logic_vector(4 downto 0);      --to read from and write to
95   signal      read_3 : std_logic_vector(4 downto 0);
96
97
98
99   ---- Control Unit-------
100  signal  instruction : std_logic_vector(24 downto 0);
101
102
103
104  --- ALU signals-----
105
106  signal      data_in_rs1 :  std_logic_vector(127 downto 0);
107  signal      data_in_rs2 :  std_logic_vector(127 downto 0);
108  signal      data_in_rs3 : std_logic_vector(127 downto 0);
109  signal      ALU_control : std_logic_vector(4 downto 0);
110
111
112
113
114
115  -----ID/EX Buffer-----
116          ---For Buffer---
117  signal      data_of_rs1 : std_logic_vector(127 downto 0);
118  signal      data_of_rs2 :  std_logic_vector(127 downto 0);
119  signal      data_of_rs3 : std_logic_vector(127 downto 0);
120  signal      control_for_ALU: std_logic_vector(4 downto 0);
121  signal      wb_control_rd_input : std_logic_vector(5 downto 0);
122
123
124
125  signal          index : std_logic_vector(127 downto 0);
126  signal          signExtend : std_logic_vector(127 downto 0);
127
128  signal          mux_control_1 : std_logic;
129  signal          mux_control_2 : std_logic;
130  signal          mux_control_Z : std_logic;
131
132
133  signal      index_out : std_logic_vector(127 downto 0);
134  signal      signExtend_out : std_logic_vector(127 downto 0);
135
136  signal      mux_control_1_out : std_logic;
137  signal      mux_control_2_out : std_logic;
138  signal      mux_control_Z_out : std_logic;
139
140
141
```

```
142
143  ----------EX/WB Buffer-------------
144
145  signal      ALU_computation : std_logic_vector(127 downto 0);
146  signal      WB_input : std_logic_vector(5 downto 0);
147
148
149          ----WriteBackData
150  signal      data_WB_out: std_logic_vector(127 downto 0);
151  signal      WB_register_rd: std_logic_vector(4 downto 0);
152
153          --writeBackControl---
154  signal      register_write_inter :std_logic;
155
156
157
158  ----------IF Register-------------
159
160  signal      counter : std_logic_vector(5 downto 0);
161
162
163
164
165  ---SIGNALS FOR FORWARDS UNITE----
166
167
168  signal      data_for_ALU_1 :    std_logic_vector(127 downto 0);
169  signal      data_for_ALU_2 :    std_logic_vector(127 downto 0);
170  signal      data_for_ALU_3 :    std_logic_vector(127 downto 0);
171
172  signal      mux_out_A :  std_logic_vector(127 downto 0);
173  signal      mux_out_B :  std_logic_vector(127 downto 0);
174  signal      mux_out_Z :  std_logic_vector(127 downto 0);
175
176  signal      rs2 : std_logic_vector(127 downto 0);
177  signal      rs2_out: std_logic_vector(127 downto 0);
178
179
180
181
182  constant period : time :=100 ns;
183
184  begin
185
186
187      U_IFRegister : entity IF_Reg
188          port map (clk=> clk , rset_bar => rset_bar, counter => counter);
189
190
191      U_Forward: entity DATA_FW
192          port map(rset_bar => rset_bar, data_of_rs1=> data_of_rs1,data_of_rs2 => data_of_rs2,
     data_of_rs3 => data_of_rs3,
193          register_num_1 => read_1, register_num_2 => read_2,register_num_3 => read_3,
     address_WB_of_rd => WB_input,  ALU_calculation_for_FW => ALU_computation,
194          fw_for_registers1 => data_in_rs1, fw_for_registers2 => data_in_rs2, fw_for_registers3
     => data_in_rs3
195
196          );
197
198
199      U_INSTRICTION_BUFFER: entity instructionBuffer
200          port map( rset_bar => rset_bar,  counter_in => counter, instructionOut => data_in);
201
202      U_IFID: entity IF_ID
203          port map(clk => clk, rset_bar => rset_bar, data_in => data_in , read_rs1 =>read_1,
204          read_rs2 =>read_2,read_rs3 =>read_3, signExtend => data_in_ext, control_in =>
     instruction);      ----- Change Read rd , should come for WB stage???
205
206      U_RegisterFile : entity RegisterFile
207          port map(rset_bar => rset_bar, read_1 => read_1, read_2 => read_2, read_3 => read_3,
     register_write => register_write_inter, data_in => data_WB_out,
```

```
208                  reg_num => WB_register_rd, out_1 => data_of_rs1, out_2=> data_of_rs2, out_3
    => data_of_rs3);
209
210
211     U_Control : entity Control
212     port map (rset_bar => rset_bar, instruction => instruction, control_out => control_for_ALU
    , control_out_WB => wb_control_rd_input,
213                  mux_select_1 => mux_control_1,mux_select_2 =>mux_control_2, mux_select_Z =>
    mux_control_z );
214
215
216     U_SignExtend_Index: entity signIndex

217     port map(rset_bar => rset_bar, data_in_ext => data_in_ext, index => index,
    signExtendOutput => signExtend, rs2 => rs2 );
218
219
220      U_ID_EX_BUFFER: entity ID_EX
221        port map(clk => clk, rset_bar => rset_bar, data_of_rs1 => data_in_rs1,data_of_rs2 =>
    data_in_rs2,data_of_rs3 =>data_in_rs3, control_for_ALU => control_for_ALU,
222                  wb_control_rd_input => wb_control_rd_input, wb_control_rd => WB_input,
223                  data_of_rs1_ALU => data_for_ALU_1, data_of_rs2_ALU => data_for_ALU_2,
    data_of_rs3_ALU =>data_for_ALU_3, control_for_ALU_out => ALU_control,
224                  index => index, signExtend => signExtend, mux_control_1 => mux_control_1,
    mux_control_2 => mux_control_2, index_out => index_out,
225                  signExtend_out => signExtend_out, mux_control_1_out => mux_control_1_out,
    mux_control_2_out => mux_control_2_out, rs2_c => rs2, rs2_out => rs2_out,
226                  mux_control_Z => mux_control_z, mux_control_Z_out => mux_control_Z_out
227
228                  );
229
230     U_MUX_A : entity mux_2to1
231         port map(rset_bar => rset_bar, data_rs => data_for_ALU_2, data_buff => mux_out_Z, cs
    => mux_control_1_out , mux_out => mux_out_A);
232
233     U_MUX_B : entity mux_2to1
234         port map(rset_bar => rset_bar, data_rs => data_for_ALU_3, data_buff => signExtend_out,
    cs => mux_control_2_out , mux_out => mux_out_B);
235
236
237
238     U_MUX_Z : entity mux_2to1
239         port map(rset_bar => rset_bar, data_rs => index_out, data_buff => rs2_out, cs =>
    mux_control_Z_out , mux_out => mux_out_Z);
240
241
242     U_Multimedia_ALU : entity ALU
243         port map(rset_bar => rset_bar, data_in_rs1 => data_for_ALU_1, data_in_rs2 => mux_out_A
    , data_in_rs3 => mux_out_B,
244         ALU_control => ALU_control, ALU_out => ALU_computation);
245
246
247         U_EX_WB_buffer: entity EX_WB
248           port map (clk=> clk, rset_bar => rset_bar, ALU_computation => ALU_computation,
    WB_input => WB_input, data_WB_out => data_WB_out, WB_register_rd => WB_register_rd,
249                     register_write => register_write_inter);
250
251     process
252
253
254
255
256
257             ---For File Input SETUP----
258     File outfile: text;
259     --constant filename_write : string :="Results.txt";
260     variable f_status:  FILE_OPEN_STATUS;
261     Variable L : Line;
262     Variable L_A: Line;
263     Variable L_B: Line;
264
```

```
265      variable j : integer :=0;
266      variable b: std_logic_vector(24 downto 0);
267
268      variable msg : string(442 downto 1);
269
270      variable instructuionW: string(25 downto 1);
271      variable register_A: string(5 downto 1);
272      variable register_B: string(5 downto 1);
273      variable register_C: string(5 downto 1);
274
275      variable content_A: string(127 downto 1);
276      variable content_B: string(127 downto 1);
277      variable content_C: string(127 downto 1);
278      variable ALU_CalculationW: string(127 downto 1);
279      variable Final_data_Out: string(127 downto 1);
280      variable destination_R : string(5 downto 1);
281
282
283
284
285
286      ----------------------------
287      variable max : integer := 2**(32-1)-1;
288      variable min : std_logic_vector(23 downto 0):=(others=>'1');
289
290
291
292
293
294      begin
295
296
297
298          ---FILE Write---
299          file_open(f_status,outfile,"Results.txt",write_mode);
300          write(L,string'("::::Instructions::::       -<rs3>- -<r2>- -<rs1>-   ::Content::
     -<r3>- -<r2>- -<r1>-  ::Control::   ::ALU_Calculation::   :::FINAL_DATA:::    -<rd>- "));
301          writeLine(outfile,L);
302          write(L,string'("====================   ============================   ===========
     ==================  ===========   ==================   ================    ====== "));
303          writeLine(outfile,L);
304          write(L,string'("
                                                                  "));
305          writeLine(outfile,L);
306          write(L,string'("
                                                                  "));
307          writeLine(outfile,L);
308
309
310
311
312      ----------------------------
313
314
315          rset_bar <= '0', '1' after 90ns;
316          clk <='0';
317
318
319          for i in 0 to 512 loop
320
321              wait for period/2;
322
323              if (i mod 2 = 0) then
324              msg := str(data_in) & " "& str(read_3) & " " &  str(read_2) & " "& str(read_1) & "
     " & str(data_of_rs1) & " "& str( control_for_ALU) & " " & str( ALU_computation) & " "& str(
     data_WB_out) & " " & str(WB_register_rd);
325              write(L,msg);
326              writeLine(outfile,L);
327              end if;
328
329              clk <= not clk;
```

File: c:/My_Designs/SMID/PipeLine SMID MultiMedia/src/Top_Level_TB.vhd (/Top_Level)

```
330           end loop;
331
332           file_close(outfile);
333
334       wait;
335       end process;
336
337
338
339
340   end behavioral_TB;
341
```

Data and Results (LI and R4-Format) :

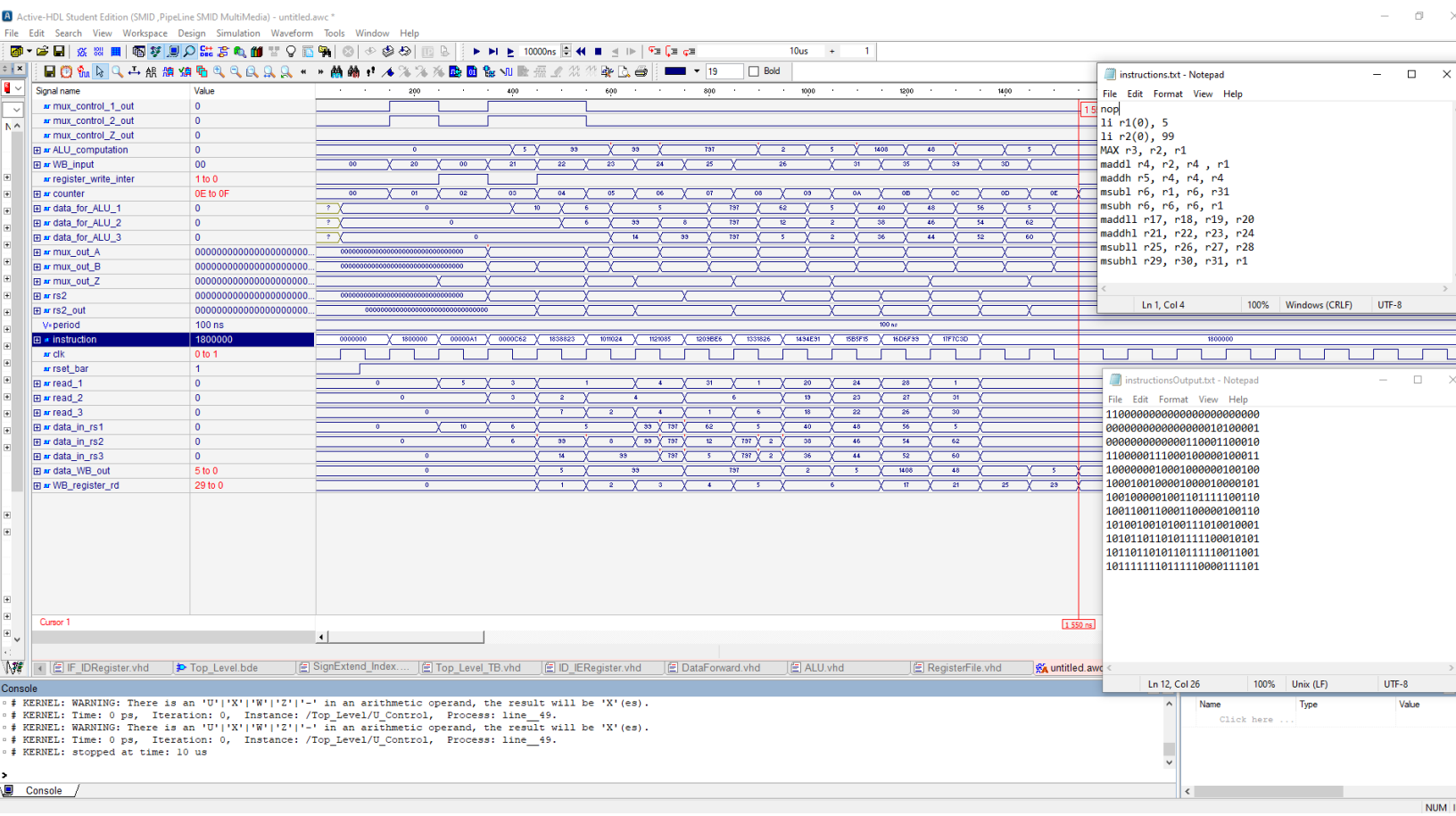


Figure 2: Shows LI and R4 format instruction results in waveforms.

We know, Load Immediate has following instruction format,

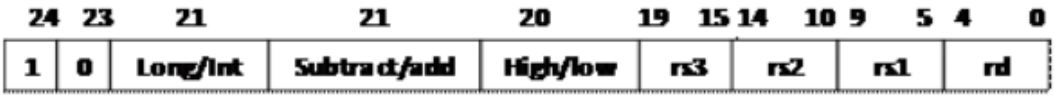
4.1 Load Immediate

Objective of Load Immediate instruction is to load a 16-bits value encoded in instruction (20 downto 5) , intro Register , Rd. This 16-bit immediate value will go into any-one-16-bit field of Register, rd, which is denoted by Load index (23-21). Since, there are 8-16-bits section in 128-bit data register, index bit 0-7 must select any one of this field. From the top-level schematic of this computer architecture, we can see additional components between Instruction Decode and Execution stage, which is designed to support this LI format instruction. Component, Sign Extend, designed using combinational logic, uses information received from Load Index and 16-bit Immediate field and extend the bit fields to a 128-bit Register value with 0 extends. The purpose of this extension is to interface with ALU component, which only supports 3, 128-bit data input. Using index value, ALU replaces a 0 valued 128-bit register with 16-bit immediate

in corresponding half word section. This data in then put on Write Back stage to be written to Register File denoted with Rd. Mux support is added after Execution stage to select signed extend Data instead of Data from Registers. This ensures correct data is applied for ALU calculation.

From Figure 2, we can see from *instructions.txt* , that there is 2 load immediate instructions after NOP instructions. 1st LI denotes, load value 5 into low half word field of Register #1, and 2nd loads value 99 into Register #2. Binary format of these instructions is also shown in the figure above. NOP instruction was used to flush out any unintended signal changes at the beginning of the simulation cycle. Therefore, instruction for Load Immediate is fetched at 2nd clock, results shows at Write Back stage at 5th clock, (4 cycles). At 5th clock we see, Write Back register is Register #1 and data to be written denotes value 5. Which is exactly what is expected using load instruction. Similarly, 2nd Load Immediate shows at 6th clock (fetched at 3rd clock inclusive), with value 99, write destination is 2. Load Immediate format is working as intended.

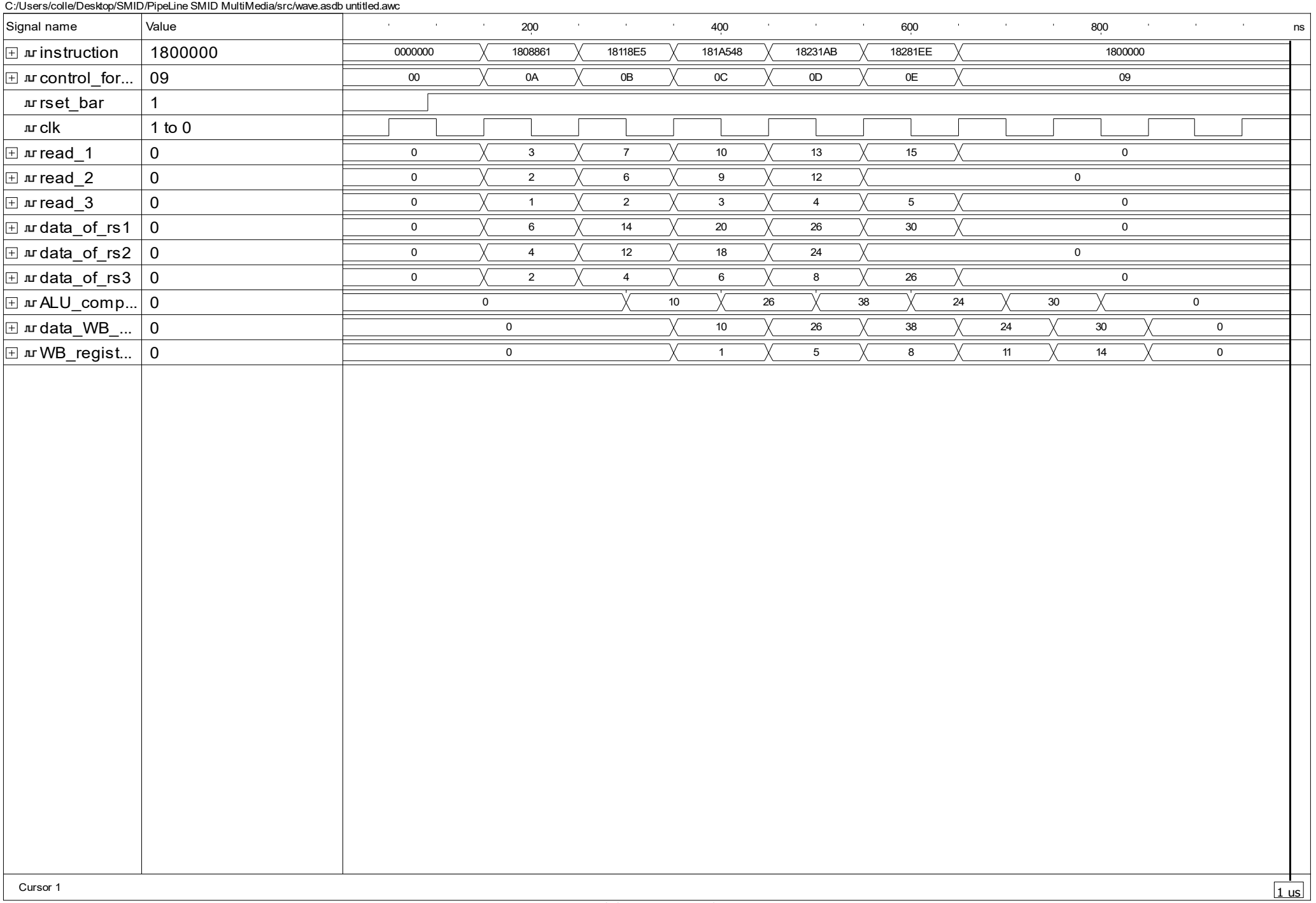
4.2 Multiply-Add and Multiply-Subtract R4-Instruction Format



Next few instructions tested using waveforms, (also written to text file) is related to R4 – format described above. We use four Registers to perform desire calculation in R4-format Instruction. Bit Fields 19 to 0 denotes corresponding registers, of which 4 to 0 is destination Register. Upper bit fields denote which type of function to use. Data for calculation is received from Execution Buffer, which in turn gets it from Register file. Mux selects form R4-format is always 0 , because intended data will need to come form Register File and not signed extended instructions fields.

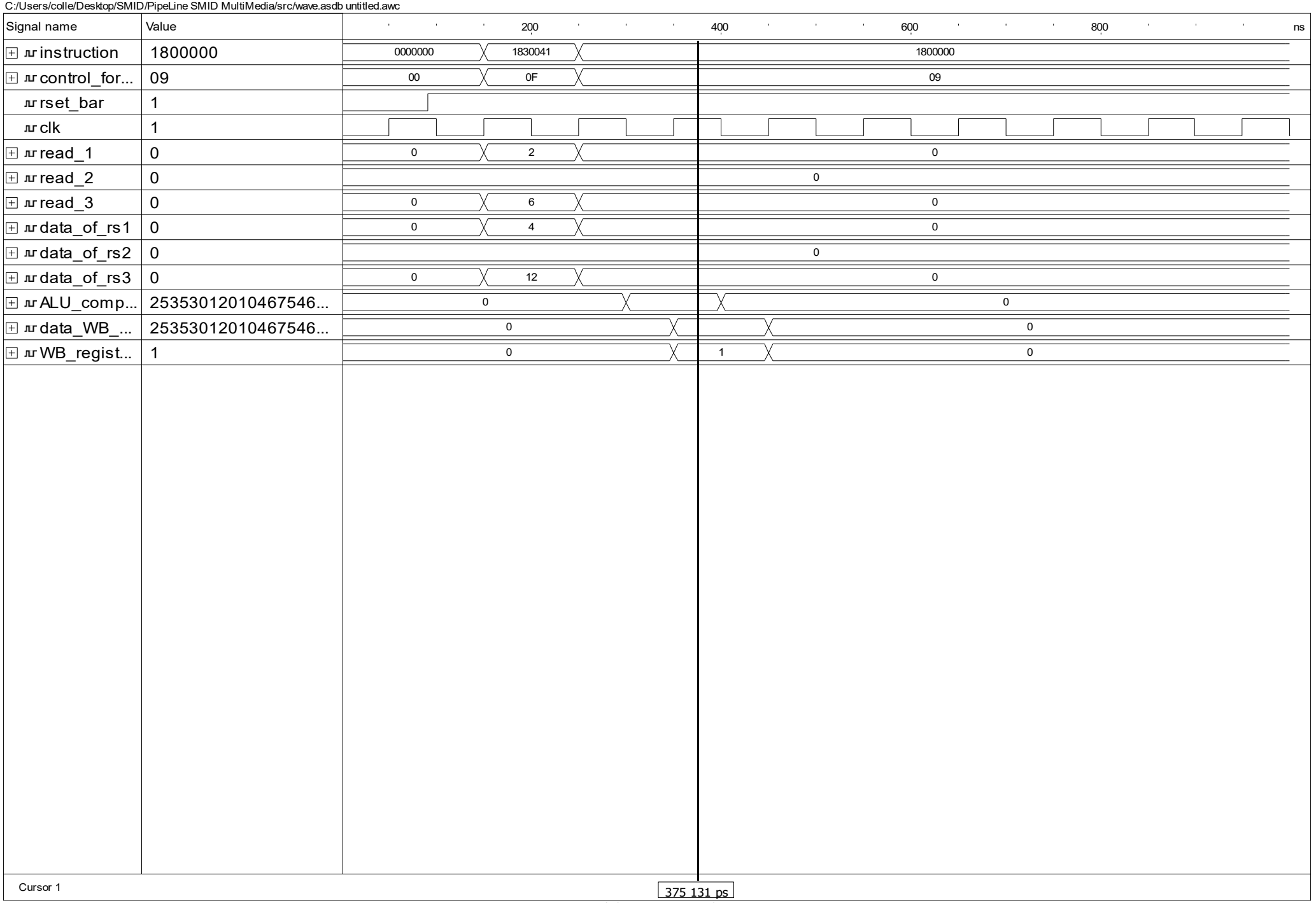
VHDL description of how each function is performed in file *ALU.vhd* . A function r4format, is described in this file, which takes appropriate fields of data and calculates desired operation. Next instruction following MAX from above figure is a R4-format Instruction, MADDL R4, R2, R4 , R1 . This instruction reads, simply, Multiply content of R2 and R4 and Add the results to content of R1, lastly place the result in R4. The letter, “L” , in MADDL , denotes multiply operation is done using low 16-bit fields of Register R2 and Register R4. Every Memory Register is initialized at the beginning of simulation with content i * 2 , where, i , is index loops from 0 to 31 . Therefore, content of R4 is 4 *2 = 8. However, content of R2 and R1 is changed using Load Immediate instruction to value 99 and 5. Therefore, low 16-bit fields are used to store these numbers in binary. Therefore, results should read , 99 * 8 + 5 = 797 and store to Register R4. Indeed, this is the results we get on 8th clock of the simulation. Next few instructions are also calculated using R4-format descriptions and results matched against waveforms. Expected results corresponds simulation data. Instruction MSUBH R6, R6, R6 , R1 , requires data forward from previous instruction. As expected, results is 5 - 0 * 0 = 5, we see data of 5 on Write Back stage with destination Register R6. Therefore, R4format is working as intended.

C:/Users/colle/Desktop/SMID/PipeLine SMID MultiMedia/src/wave.asdb untitled.awc

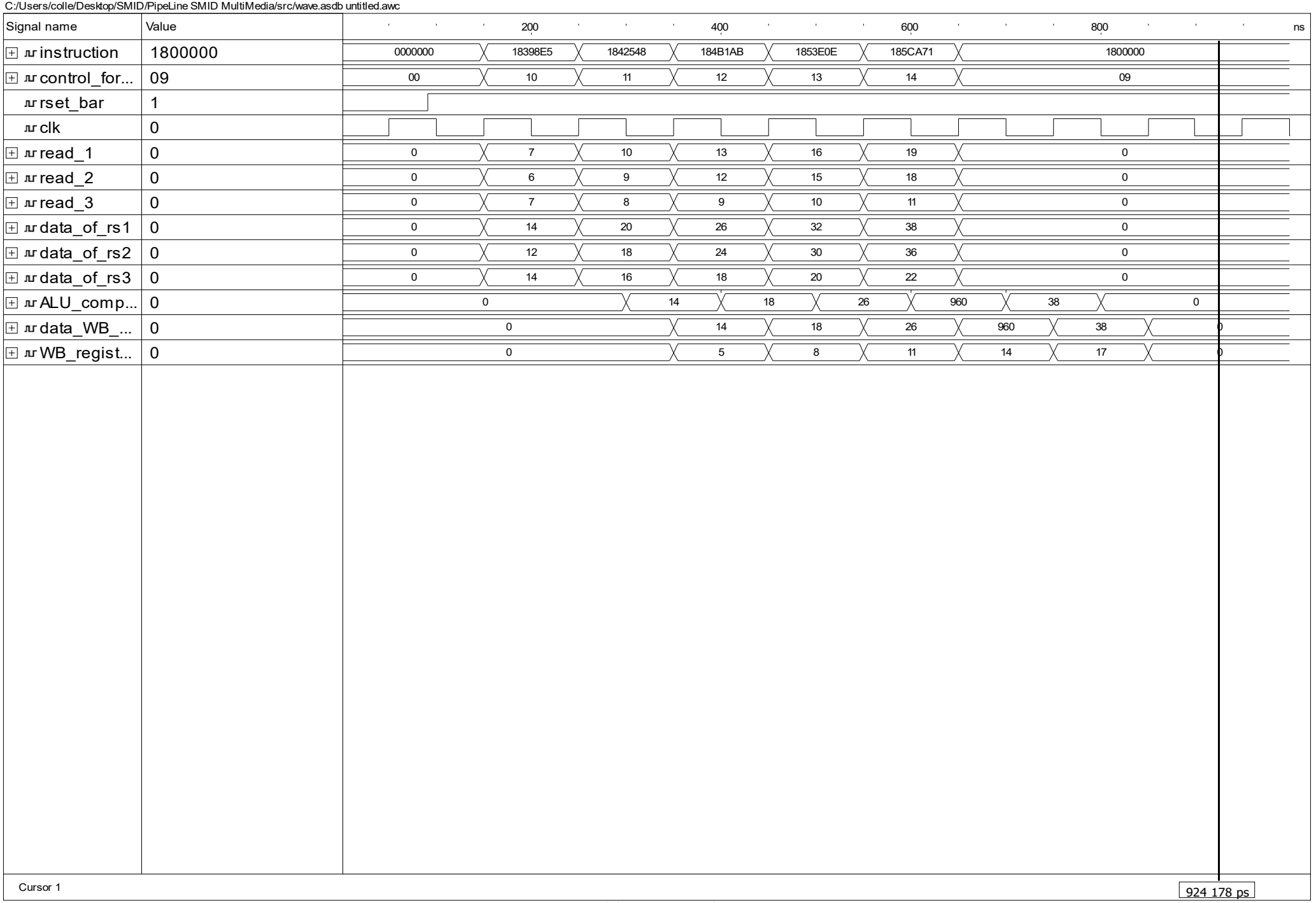


1/1 (0 ps - 1 022 259 ps)

C:/Users/colle/Desktop/SMID/PipeLine SMID MultiMedia/src/wave.asdb untitled.awc

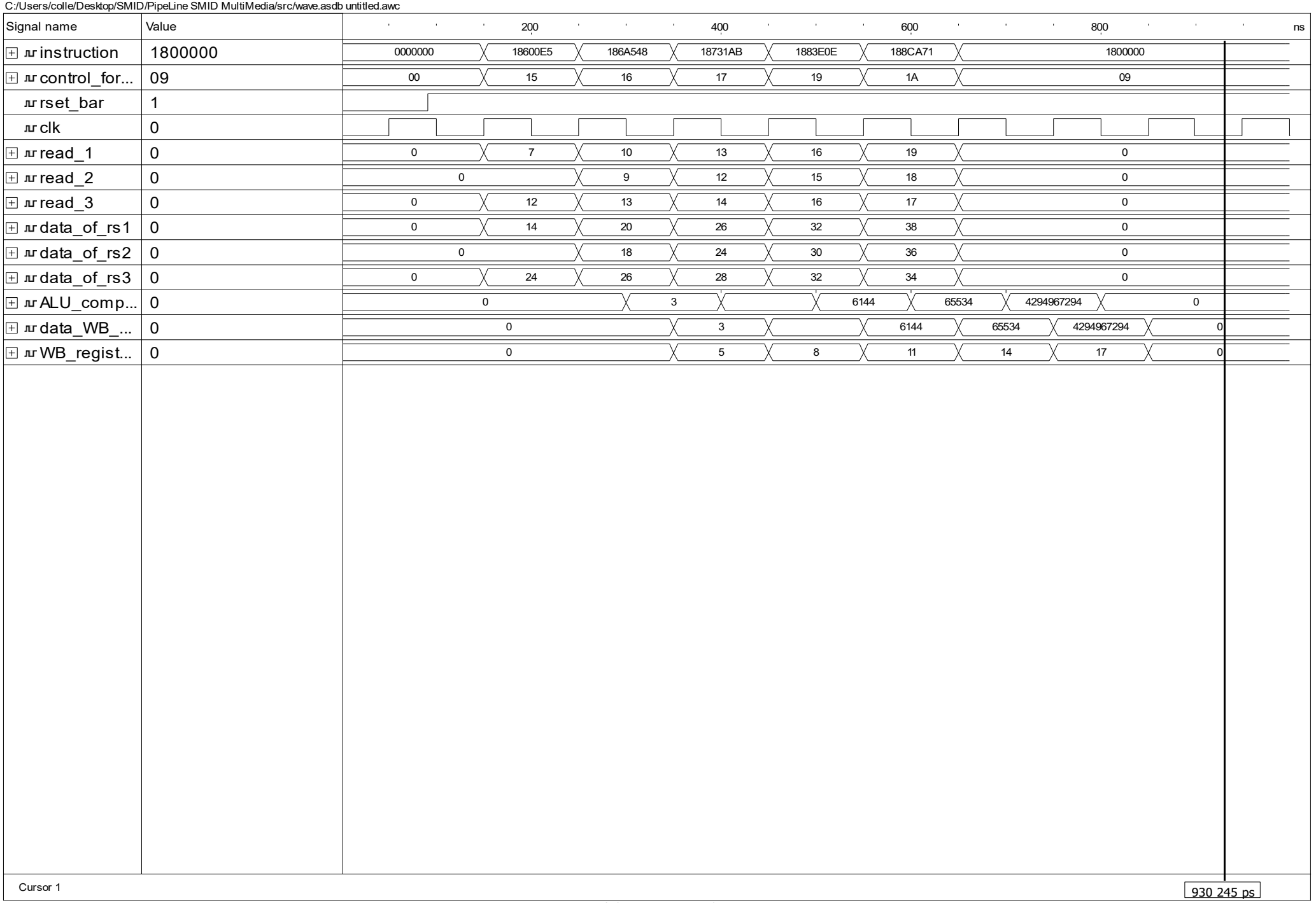


C:/Users/colle/Desktop/SMID/PipeLine SMID MultiMedia/src/wave.asdb untitled.awc



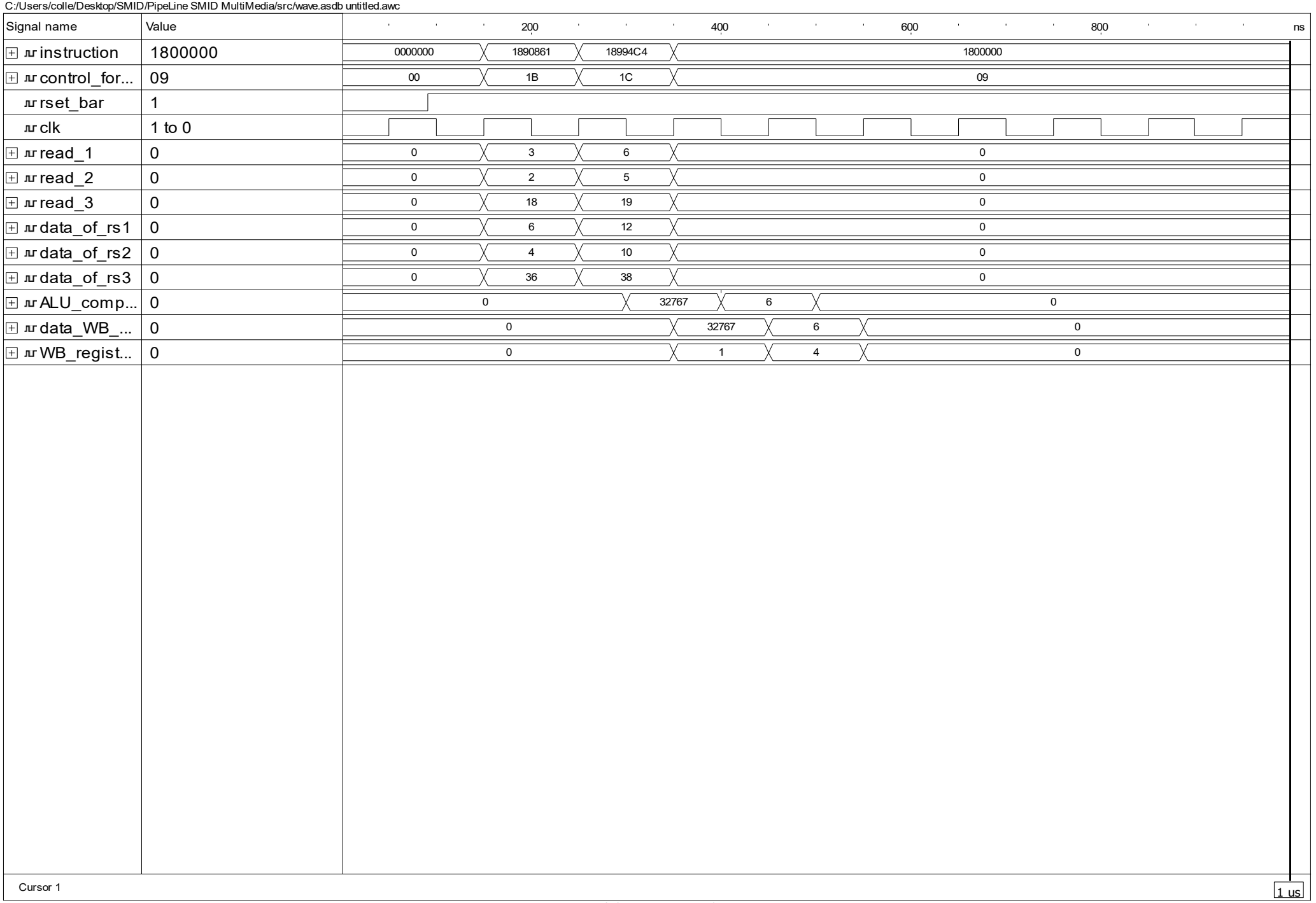
1/1 (0 ps - 1 022 259 ps)

C:/Users/colle/Desktop/SMID/PipeLine SMID MultiMedia/src/wave.asdb untitled.awc



1/1 (0 ps - 1 022 259 ps)

C:/Users/colle/Desktop/SMID/PipeLine SMID MultiMedia/src/wave.asdb untitled.awc



1/1 (0 ps - 1 022 259 ps)

Through a lot of struggles, the project was a success. Each individual component worked as they expected. Testbenches were written for all of the components to verify their functionality. The source-code for the individual components and testbenches as well as their simulated waveforms and all included. Combining all of the components in a structural fashion yielded the system. The system is fully functional with forwarding. The system works for all twenty-nine instructions without any error.