Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	**Computer Organization and Architecture**
Course Number:	**COE608**
Semester/Year (e.g.F2016)	**Winter 2023**

Instructor:	**Demetres Kostas**

Assignment/Lab Number:	*4b*
Assignment/Lab Title:	*Data Memory Module*

Submission Date:	*Wednesday March 15 2023*
Due Date:	*Wednesday March 15 2023 3:00pm*

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Patel	**Stuti**		**04**	**S.P**

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

VHDL CODES

Register 32

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity register32 is
port(
     d         : in std_logic_vector(31 downto 0);
     ld        : in std_logic;
     clr       : in std_logic;
     clk       : in std_logic;
     Q         : out std_logic_vector(31 downto 0)

      );
end register32;
```

Add

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add is
port (A    : in std_logic_vector(31 downto 0);
B    : out std_logic_vector(31 downto 0)
);
end add;

architecture Behavior of add is
begin
B <= A + 4;
end Behavior;
```

Mux2to1

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is
port (s    :  in std_logic;
w0, w1 :  in std_logic_vector(31 downto 0);
f      :  out std_logic_vector(31 downto 0)
);
end mux2to1;

architecture Behavior of mux2to1 is
begin
with s select
f <= w0 when '0',
 w1 when others;
end Behavior;
```

PC

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pc is
port(
clr : in    std_logic;
clk : in    std_logic;
ld  : in    std_logic;
inc : in    std_logic;
d   : in    std_logic_vector(31 downto 0);
q   : out    std_logic_vector(31 downto 0)
);
end pc;

architecture Behavior of pc is
component add
port(
A   : in  std_logic_vector(31 downto 0);
B   : out std_logic_vector(31 downto 0)
);
end component;

component mux2to1
port (s   :  in std_logic;
w0, w1 :  in std_logic_vector(31 downto 0);
f      :  out std_logic_vector(31 downto 0)
);
end component;

component register32
port (
d   : in std_logic_vector(31 downto 0);
ld   : in std_logic;
clr   : in std_logic;
clk   : in std_logic;
Q   : out std_logic_vector(31 downto 0)
);
end component;

signal add_out : std_logic_vector(31 downto 0);
signal mux_out : std_logic_vector(31 downto 0);
signal q_out   : std_logic_vector(31 downto 0);
begin
add0: add port map(q_out, add_out);
mux0: mux2to1 port map(inc, d, add_out, mux_out);
reg0: register32 port map (mux_out, ld, clr, clk, q_out);
q <= q_out;
end Behavior;
```

Fulladd

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity fulladd is
    port(
        Cin, x, y   : in   std_logic;
        s, Cout     : out  std_logic
       );
end fulladd;

architecture Behavior of fulladd is
begin
        s <= x xor y xor Cin;
        Cout <= (x and y) or (Cin and x) or (Cin and y);
end Behavior;
```

Adder4

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity adder4 is
port(
    Cin     :  in std_logic;
    X,Y     :  in std_logic_vector(3 downto 0);
    S     :  out  std_logic_vector(3 downto 0);
    Cout :  out  std_logic
       );
end adder4;

architecture Behaviour of adder4 is
    component fulladd
    port(
    Cin, x, y : in  std_logic;
    s, Cout   : out std_logic
    );
    end component;

    signal C : std_logic_vector(1 to 3);
begin
    stage0: fulladd port map(Cin,  X(0), Y(0), S(0), C(1));
    stage1: fulladd port map(C(1), X(1), Y(1), S(1), C(2));
    stage2: fulladd port map(C(2), X(2), Y(2), S(2), C(3));
    stage3: fulladd port map(C(3), X(3), Y(3), S(3), Cout);
end Behaviour;
```

Adder16

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity adder16 is
port(
        Cin     :  in std_logic;
        X,Y     :  in std_logic_vector(15 downto 0);
        S     :  out  std_logic_vector(15 downto 0);
        Cout :  out  std_logic
       );
end adder16;

architecture Behaviour of adder16 is
    component adder4
    port(
        Cin     :  in std_logic;
        X,Y     :  in std_logic_vector(3 downto 0);
        S     :  out  std_logic_vector(3 downto 0);
        Cout :  out  std_logic
       );
end component;

    signal C : std_logic_vector(1 to 3);
begin
    stage0: adder4 port map(Cin,  X(3 downto 0), Y(3 downto 0), S(3 downto 0), C(1));
    stage1: adder4 port map(C(1), X(7 downto 4), Y(7 downto 4), S(7 downto 4), C(2));
    stage2: adder4 port map(C(2), X(11 downto 8), Y(11 downto 8), S(11 downto 8), C(3));
    stage3: adder4 port map(C(3), X(15 downto 12), Y(15 downto 12), S(15 downto 12), Cout);
end Behaviour;
```

Adder32

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity adder32 is
port(
      Cin    :  in std_logic;
      X,Y    :  in std_logic_vector(31 downto 0);
      S    :  out  std_logic_vector(31 downto 0);
      Cout :  out  std_logic
      );
end adder32;

architecture Behaviour of adder32 is
   component adder16
   port(
      Cin    :  in std_logic;
      X,Y    :  in std_logic_vector(15 downto 0);
      S    :  out  std_logic_vector(15 downto 0);
      Cout :  out  std_logic
   );
   end component;

   signal C : std_logic;
begin
   stage0: adder16 port map(Cin,  X(15 downto 0), Y(15 downto 0), S(15 downto 0), C);
   stage1: adder16 port map(C, X(31 downto 16), Y(31 downto 16), S(31 downto 16), Cout);
end Behaviour;
```

Alu

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity ALU is
   port(
        a    :  in std_logic_vector(31 downto 0);
        b    :  in std_logic_vector(31 downto 0);
        op   :  in std_logic_vector(2 downto 0);
        result : out  std_logic_vector(31 downto 0);
        zero : out std_logic;
        cout : out std_logic
        );
end ALU;

architecture Behaviour of ALU is
   component adder32
   port(
      Cin    :  in std_logic;
      X,Y    :  in std_logic_vector(31 downto 0);
      S    :  out  std_logic_vector(31 downto 0);
      Cout :  out  std_logic );

   end component;

   signal result_s: std_logic_vector(31 downto 0):= (others => '0');
```

```vhdl
   signal result_s: std_logic_vector(31 downto 0):= (others => '0');
   signal result_add: std_logic_vector(31 downto 0):= (others => '0');
   signal result_sub: std_logic_vector(31 downto 0):= (others => '0');
   signal cout_s  : std_logic := '0';
   signal cout_add  : std_logic := '0';
   signal cout_sub  : std_logic := '0';
   signal zero_s    : std_logic:= '0';

begin
     add0 : adder32 port map (op(2), a, b, result_add,cout_add);
     sub0 : adder32 port map (op(2), a, not b, result_sub,cout_sub);

     process (a, b, op)
     begin
       case (op) is
         when "000" =>
            result_s<= a and b;
            cout_s <= '0';
         when "001" =>
            result_s<= a or b;
            cout_s <= '0';
         when "010" =>
            result_s<= result_add;
            cout_s <= cout_add;
         when "011" =>
```

```
                result_s<= b;
                cout_s <= '0';
            when "110" =>
                result_s<= result_sub;
                cout_s <= cout_sub;
            when "100" =>
                result_s<= a(30 downto 0) & '0';
                cout_s <= a(31);
            when "101" =>
                result_s<= '0' & a(31 downto 1);
                cout_s <= '0';
            when others =>
                result_s <= a;
                cout_s <= '0';
            end case;

            case(result_s) is
                when (others => '0') =>
                    zero_s <= '1';
                when others =>
                    zero_s <= '0';
            end case;
        end process;

        result <= result_s;
        cout <= cout_s;
        zero <= zero_s;
    end Behaviour;
```

Data_mem

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity data_mem is
port(
    clk             : in std_logic;
    addr            : in unsigned(7 downto 0);
    data_in         : in std_logic_vector(31 downto 0);
    wen             : in std_logic;
    en              : in std_logic;
    data_out        : out std_logic_vector(31 downto 0)
    );
end data_mem;

architecture Behavior of  data_mem is
    type RAM is array (0 to 255) of std_logic_vector(31 downto 0);
    signal DATAMEM : RAM;
begin
    process(clk, en, wen)
    begin
        if(clk'event and clk='0') then
            if (en = '0') then
                data_out <= (others => '0');
            else
                if (wen = '0') then
                    data_out <= DATAMEM (to_integer(addr));
                end if;
```

```
            end if;
            if (wen = '1') then
                DATAMEM (to_integer(addr)) <= data_in;
                data_out <= (others => '0');
            end if;
        end if;
    end if;
    end process;
end Behavior;
```

LZE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std. all;

entity LZE is
    port (
        LZE_in : in std_logic_vector(31 downto 0);
        LZE_out : out std_logic_vector(31 downto 0)
    );
end entity;

architecture Behavior of LZE is
signal zeros: std_logic_vector(15 downto 0) := (others => '0');
begin
    LZE_out <= zeros & LZE_in(15 downto 0);
end Behavior;
```

UZE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UZE is
   port(
      UZE_in   :  in std_logic_vector(31 downto 0);
      UZE_out  :  out std_logic_vector(31 downto 0)
   );
end entity;

architecture Behavior of UZE is
   signal zeros: std_logic_vector(15 downto 0) := (others => '0');
begin
   UZE_out <= UZE_in(15 downto 0) & zeros;
end Behavior;
```

RED

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RED is
port (
    RED_in   :  in std_logic_vector(31 downto 0);
    RED_out  :  out unsigned(7 downto 0)
);
end entity;

architecture Behavior of RED is
begin
   RED_out <= unsigned (RED_in(7 downto 0));
end Behavior;
```

Mux4to1

```
library ieee;
use ieee.std_logic_1164.all;

entity mux4to1 is
   port(
      s              :  in  std_logic_vector(1 downto 0);
      X1, X2, X3, X4 :  in  std_logic_vector(31 downto 0);
      f              :  out std_logic_vector(31 downto 0)
   );
end mux4to1;

architecture Behavior of mux4to1 is
begin
   with s select
      f <=  X1 when "00",
            X2 when "01",
            X3 when "10",
            X4 when "11";
end Behavior;
```

Data_Path

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity DataPath is
    port(
        --Clock Signal
        Clk, mClk    : in std_logic;

        --Memory Signals
        WEN, EN      : in std_logic;

        --Register Control Signals (CIR and LD)
        Clr_A, Ld_A     : in std_logic;
        Clr_B, Ld_B     : in std_logic;
        Clr_C, Ld_C     : in std_logic;
        Clr_Z, Ld_Z     : in std_logic;
        ClrPC, Ld_PC    : in std_logic;
        ClrIR, Ld_IR    : in std_logic;

        --Register Outputs
        Out_A     : out std_logic_vector(31 downto 0);
        out_B     : out std_logic_vector(31 downto 0);
        Out_C     : out std_logic;
        Out_Z     : out std_logic;
        Out_PC    : out std_logic_vector(31 downto 0);
        Out_IR    : out std_logic_vector(31 downto 0);

        --Special Inputs to PC
        Inc_PC    : in std_logic;

        --Address and Data Bus signals for debugging
        ADDR_OUT     : out std_logic_vector(31 downto 0);
        DATA_IN      : in std_logic_vector(31 downto 0);
        DATA_BUS,
```

```vhdl
        ADDR_OUT     : out std_logic_vector(31 downto 0);
        DATA_IN      : in std_logic_vector(31 downto 0);
        DATA_BUS,
        MEM_OUT,
        MEM_IN       : out std_logic_vector(31 downto 0);
        MEM_ADDR     : out unsigned(7 downto 0);

        --Various MUX controls
        DATA_MUX     : in std_logic_vector(1 downto 0);
        REG_MUX      : in std_logic;
        A_MUX,
        B_MUX        : in std_logic;
        IM_MUX1      : in std_logic;
        IM_MUX2      : in std_logic_vector(1 downto 0);

        --ALU Operations
        ALU_Op       : in std_logic_vector(2 downto 0)
    );
end DataPath;

architecture Behavior of DataPath is

    -- Components
    -- data memory
    component data_mem is
        port(
            clk      : in  std_logic;
            addr     : in  unsigned(7 downto 0);
            data_in  : in  std_logic_vector(31 downto 0);
            wen      : in  std_logic;
            en       : in  std_logic;
            data_out : out std_logic_vector(31 downto 0)
        );
    end component;

```

```vhdl
        );
    end component;

    -- register32
    component register32 is
        port(
            d     :  in  std_logic_vector(31 DOWNTO 0);
            ld    :  in  std_logic;
            clr   :  in  std_logic;
            clk   :  in  std_logic;
            Q     :  out std_logic_vector(31 DOWNTO 0)
        );
    end component;

    -- program counter
    component pc is
        port (
            clr   :  in    std_logic;
            clk   :  in    std_logic;
            ld    :  in    std_logic;
            inc   :  in    std_logic;
            d     :  in    std_logic_vector(31 downto 0);
            q     :  out   std_logic_vector(31 downto 0)
        );
    end component;

    -- lze
    component LZE is
        port (
            LZE_in : in std_logic_vector(31 downto 0);
            LZE_out : out std_logic_vector(31 downto 0)
        );
    end component;

    -- uze
    component UZE is
```

```
100
101     -- uze
102     component UZE is
103        port(
104           UZE_in   :  in std_logic_vector(31 downto 0);
105           UZE_out  :  out std_logic_vector(31 downto 0)
106        );
107     end component;
108
109     -- red
110     component RED is
111        port (
112           RED_in   :  in std_logic_vector(31 downto 0);
113           RED_out  :  out unsigned(7 downto 0)
114        );
115     end component;
116
117     -- mux2to1
118     component mux2to1 is
119        port (
120           s        : in  std_logic;
121           w0, w1   : in  std_logic_vector(31 downto 0);
122           f        : out std_logic_vector(31 downto 0)
123        );
124     end component;
125
126     -- mux4to1
127     component mux4to1 is
128        port(
129           s              :  in  std_logic_vector(1 downto 0);
130           X1, X2, X3, X4 :  in  std_logic_vector(31 downto 0);
131           f              :  out std_logic_vector(31 downto 0)
132        );
133     end component;
134
135     -- alu
```

```
133     end component;
134
135     -- alu
136     component alu is
137        port (
138           a       : in  std_logic_vector(31 downto 0);
139           b       : in  std_logic_vector(31 downto 0);
140           op      : in  std_logic_vector(2 downto 0);
141           result  : out std_logic_vector(31 downto 0);
142           cout    : out std_logic;
143           zero    : out std_logic
144        );
145     end component;
146
147     -- Signals
148     signal IR_OUT           : std_logic_vector(31 downto 0);
149     signal data_bus_s       : std_logic_vector(31 downto 0);
150     signal LZE_out_PC       : std_logic_vector(31 downto 0);
151     signal LZE_out_A_Mux    : std_logic_vector(31 downto 0);
152     signal LZE_out_B_Mux    : std_logic_vector(31 downto 0);
153     signal RED_out_Data_Mem : unsigned(7 downto 0);
154     signal A_Mux_out        : std_logic_vector(31 downto 0);
155     signal B_Mux_out        : std_logic_vector(31 downto 0);
156     signal reg_A_out        : std_logic_vector(31 downto 0);
157     signal reg_B_out        : std_logic_vector(31 downto 0);
158     signal reg_Mux_out      : std_logic_vector(31 downto 0);
159     signal data_mem_out     : std_logic_vector(31 downto 0);
160     signal UZE_IM_MUX1_out  : std_logic_vector(31 downto 0);
161     signal IM_MUX1_out      : std_logic_vector(31 downto 0);
162     signal LZE_IM_MUX2_out  : std_logic_vector(31 downto 0);
163     signal IM_MUX2_out      : std_logic_vector(31 downto 0);
164     signal ALU_out          : std_logic_vector(31 downto 0);
165     signal zero_flag        : std_logic;
166     signal carry_flag       : std_logic;
167     signal temp             : std_logic_vector(30 downto 0) := (others => '0');
168     signal out_pc_sig       : std_logic_vector(31 downto 0);
```

```
166     signal carry_flag       : std_logic;
167     signal temp             : std_logic_vector(30 downto 0) := (others => '0');
168     signal out_pc_sig       : std_logic_vector(31 downto 0);
169  begin
170     IR:   register32 port map(
171        data_bus_s,
172        Ld_IR,
173        ClrIR,
174        Clk,
175        IR_OUT
176     );
177
178     LZE_PC:  LZE port map(
179        IR_OUT,
180        LZE_out_PC
181     );
182
183     PC0:  PC port map(
184        CLRPC,
185        Clk,
186        ld_PC,
187        INC_PC,
188        LZE_out_PC,
189        --ADDR_OUT,
190        out_pc_sig
191     );
192
193     LZE_A_Mux:  LZE port map(
194        IR_OUT,
195        LZE_out_A_Mux
196     );
197
198     A_mux0:  mux2to1 port map(
199        A_MUX,
200        data_bus_s,
201        LZE_out_A_Mux,
```

```
        A_MUX,
        data_bus_s,
        LZE_out_A_Mux,
        A_Mux_out
    );

    Reg_A:   register32 port map(
        A_Mux_out,
        Ld_A,
        Clr_A,
        Clk,
        reg_A_out
    );

    LZE_B_Mux:  LZE port map(
        IR_OUT,
        LZE_out_B_Mux
    );

    B_Mux0:  mux2to1 port map(
        B_MUX,
        data_bus_s,
        LZE_out_B_Mux,
        B_mux_out
    );

    Reg_B:   register32 port map(
        B_mux_out,
        Ld_B,
        Clr_B,
        Clk,
        reg_B_out
    );

    Reg_Mux0:   mux2to1 port map(
        REG_MUX,
```

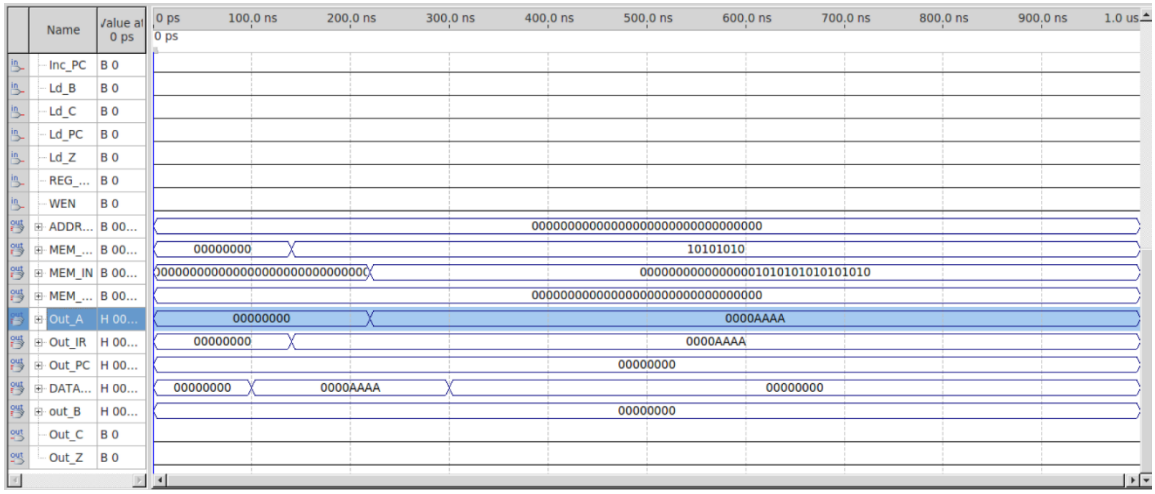
```
    Reg_Mux0:   mux2to1 port map(
        REG_MUX,
        Reg_A_out,
        Reg_B_out,
        Reg_Mux_out
    );

    RED_Data_Mem:  RED port map(
        IR_OUT,
        RED_out_Data_Mem
    );

    Data_mem0:  data_mem port map(
        mClk,
        RED_out_Data_Mem,
        Reg_Mux_out,
        WEN,
        EN,
        data_mem_out
    );

    UZE_IM_MUX1:   UZE port map(
        IR_OUT,
        UZE_IM_MUX1_out
    );

    IM_MUX1a:   mux2to1 port map(
        IM_MUX1,
        reg_A_out,
        UZE_IM_MUX1_out,
        IM_MUX1_out
    );

    LZE_IM_MUX2:   LZE port map(
        IR_OUT,
```

```
    LZE_IM_MUX2:   LZE port map(
        IR_OUT,
        LZE_IM_MUX2_out
    );

    IM_MUX2a:   mux4to1 port map(
        IM_MUX2,
        reg_B_out,
        LZE_IM_MUX2_out,
        (temp & '1'),
        (others => '0'),
        IM_MUX2_out
    );

    ALU0: ALU port map(
        IM_MUX1_out,
        IM_MUX2_out,
        ALU_Op,
        ALU_out,
        zero_flag ,
        carry_flag
    );

    DATA_MUX0:  mux4to1 port map(
        DATA_MUX,
        DATA_IN,
        data_mem_out,
        ALU_out,
        (others => '0'),
        data_bus_s
    );

    DATA_BUS <= data_bus_s;
    OUT_A <= reg_A_out;
    OUT_B <= reg_B_out;
```

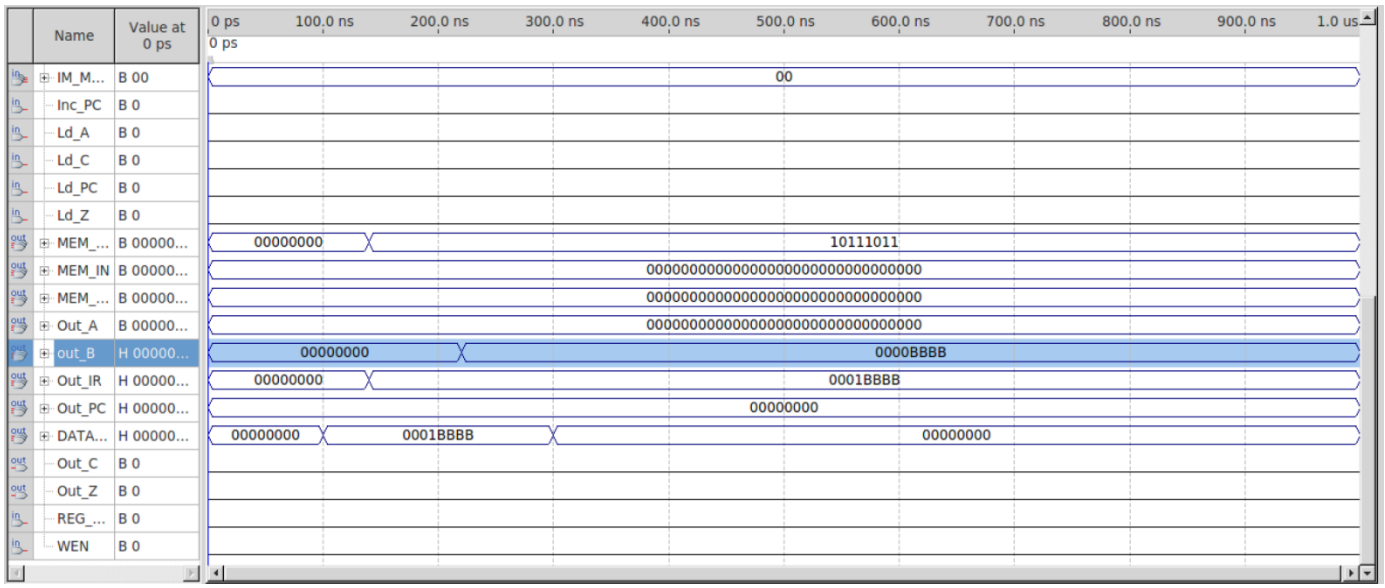
```
300     OUT_B <= reg_B_out;
301     OUT_IR <= IR_OUT;
302     ADDR_OUT <= out_pc_sig;
303     OUT_PC <= out_pc_sig;
304
305     MEM_ADDR <= RED_out_Data_Mem;
306     MEM_IN <= Reg_Mux_out;
307     MEM_OUT <= data_mem_out;
308
309
310 end Behavior;
```

Functional Simulations

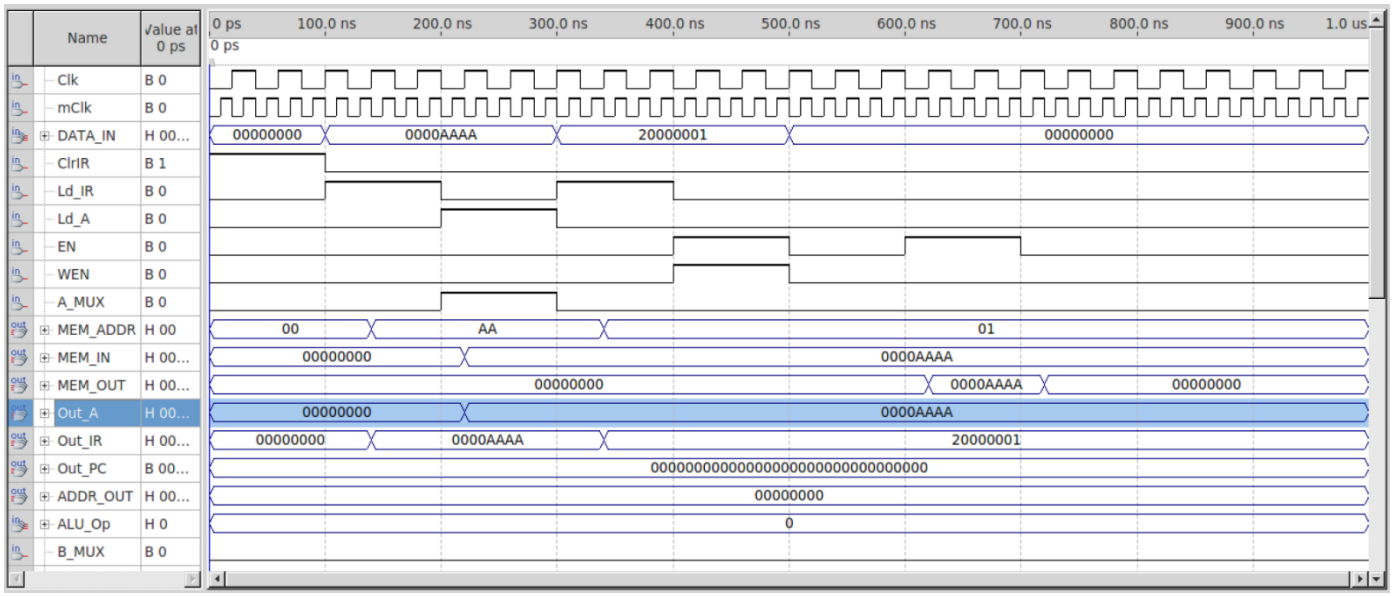
LDAI



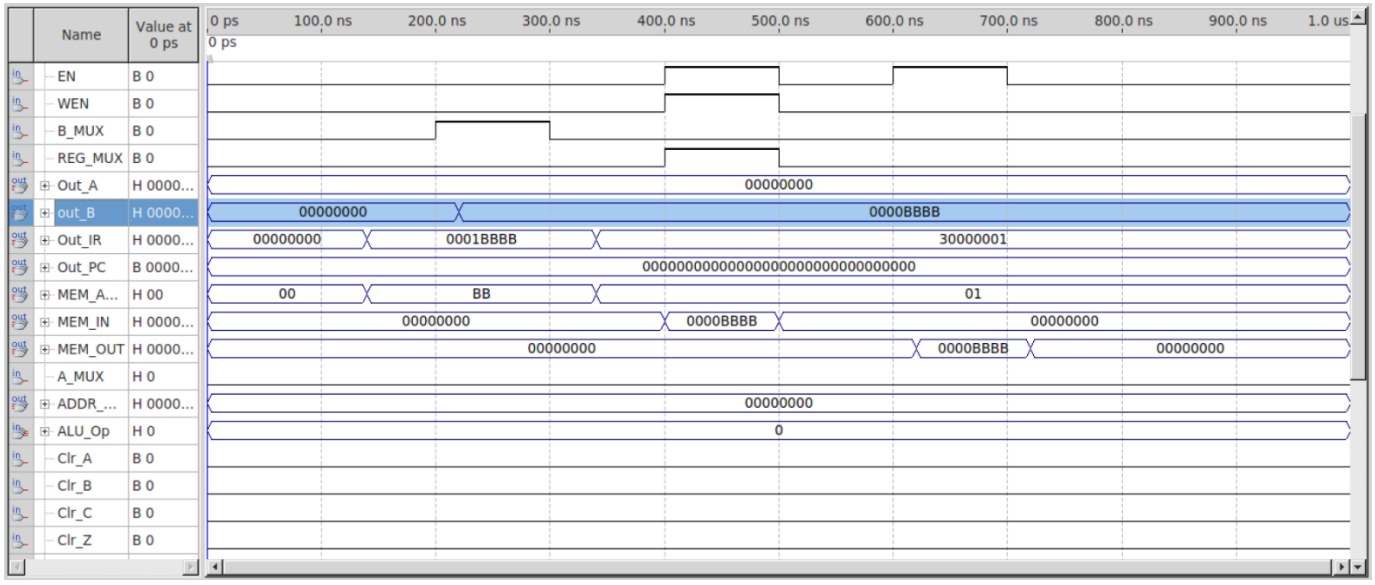
LDBI



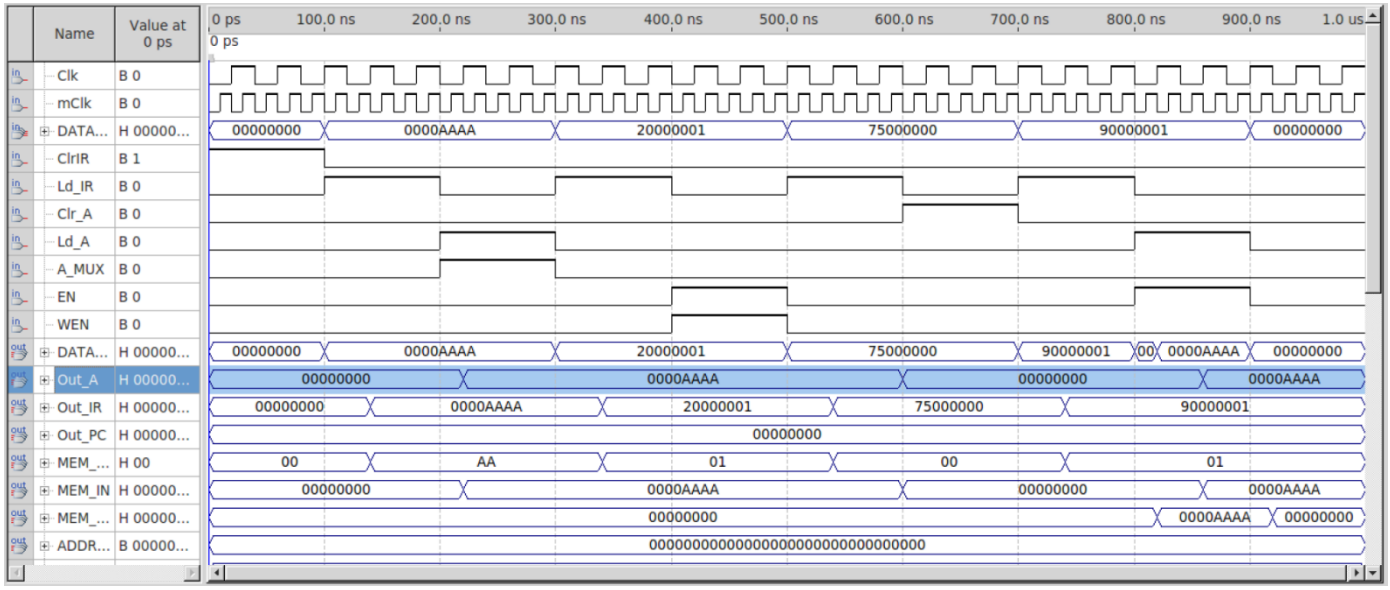
STA


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk B 0
mClk B 0
DATA_IN H 00...
00000000
0000AAAA
20000001
00000000
ClrIR B 1
Ld_IR B 0
Ld_A B 0
EN B 0
WEN B 0
A_MUX B 0
MEM_ADDR H 00
00
AA
01
MEM_IN H 00...
00000000
0000AAAA
MEM_OUT H 00...
00000000
0000AAAA
00000000
Out_A H 00...
00000000
0000AAAA
Out_IR H 00...
00000000
0000AAAA
20000001
Out_PC B 00...
00000000000000000000000000000000
ADDR_OUT H 00...
00000000
ALU_Op H 0
0
B_MUX B 0

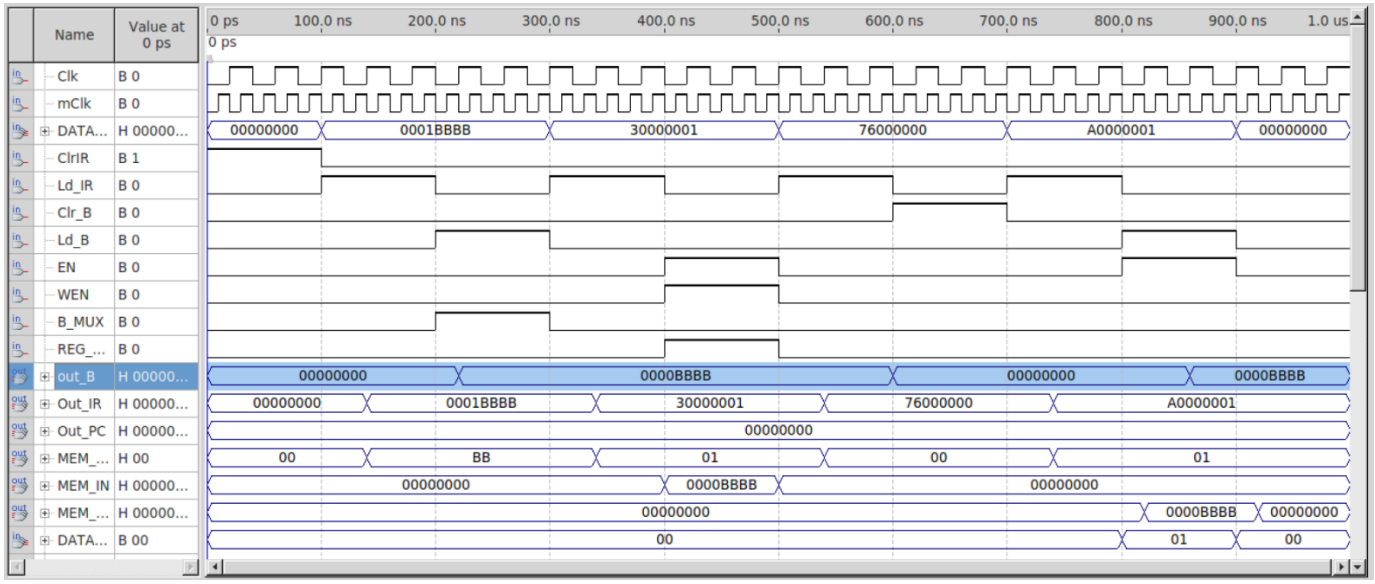
STB


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
EN B 0
WEN B 0
B_MUX B 0
REG_MUX B 0
Out_A H 0000...
00000000
out_B H 0000...
00000000
0000BBBB
Out_IR H 0000...
00000000
0001BBBB
30000001
Out_PC B 0000...
00000000000000000000000000000000
MEM_A... H 00
00
BB
01
MEM_IN H 0000...
00000000
0000BBBB
00000000
MEM_OUT H 0000...
00000000
0000BBBB
00000000
A_MUX H 0
ADDR_... H 0000...
00000000
ALU_Op H 0
0
Clr_A B 0
Clr_B B 0
Clr_C B 0
Clr_Z B 0

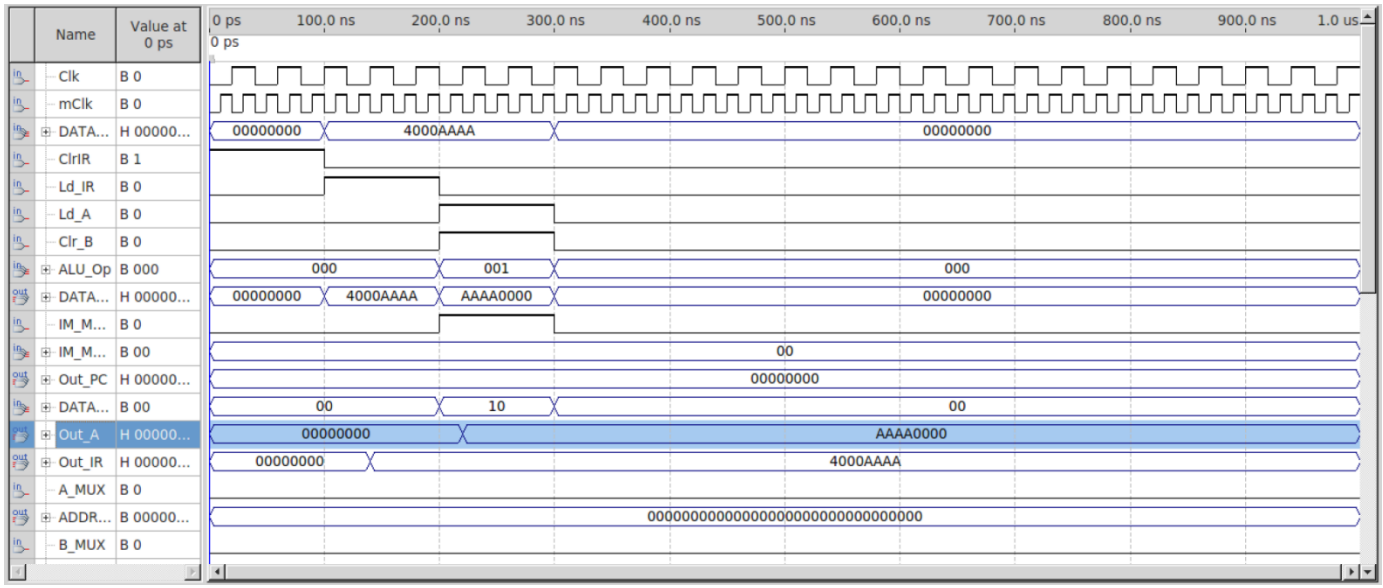
LDA


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
Clr_A
Ld_A
A_MUX
EN
WEN
Out_A
Out_IR
Out_PC
MEM_...
MEM_IN
ADDR...
00000000
0000AAAA
20000001
75000000
90000001
AA
01

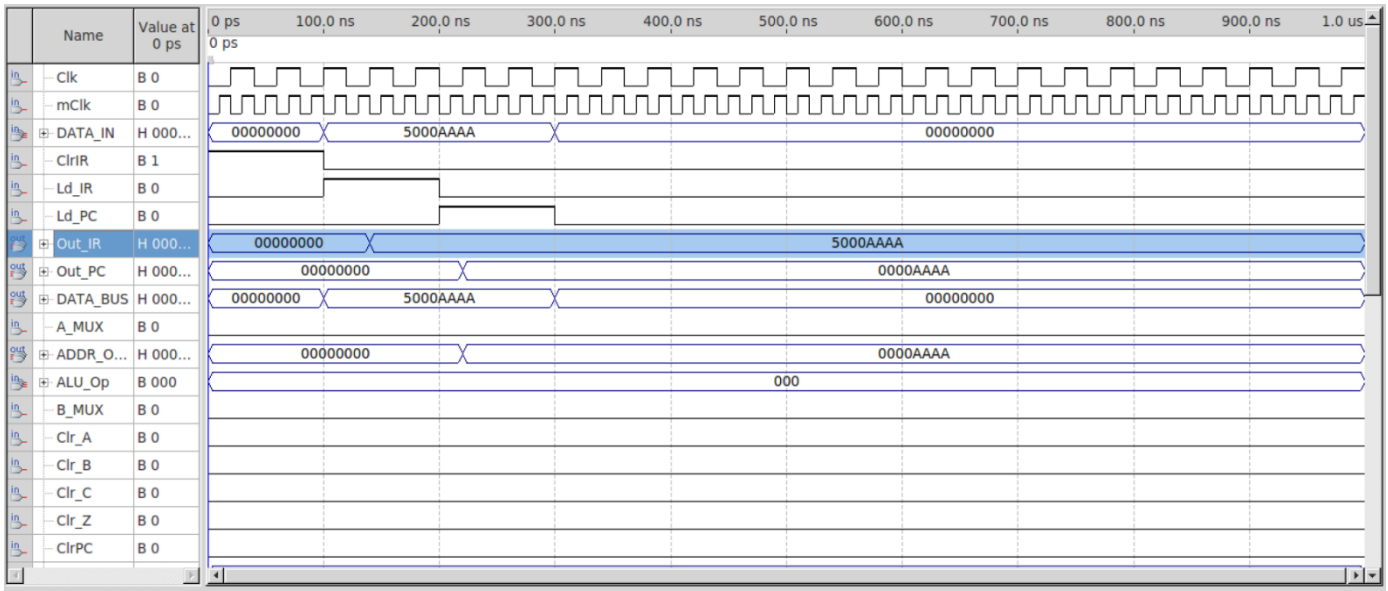
LDB


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
Clr_B
Ld_B
EN
WEN
B_MUX
REG_...
out_B
Out_IR
Out_PC
MEM_...
MEM_IN
00000000
0001BBBB
30000001
76000000
A0000001
0000BBBB
BB
01

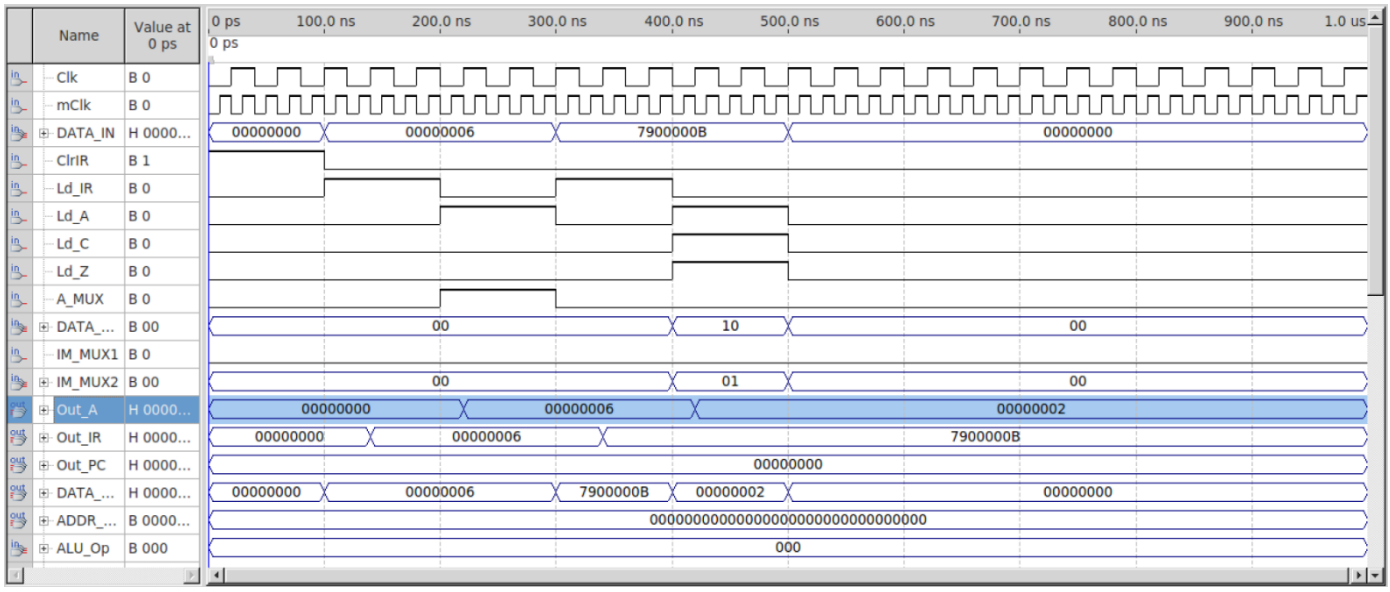
LUI


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
Ld_A
Clr_B
ALU_Op
IM_M...
Out_PC
Out_A
Out_IR
A_MUX
ADDR...
B_MUX
00000000
4000AAAA
AAAA0000
000
001
00
10
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

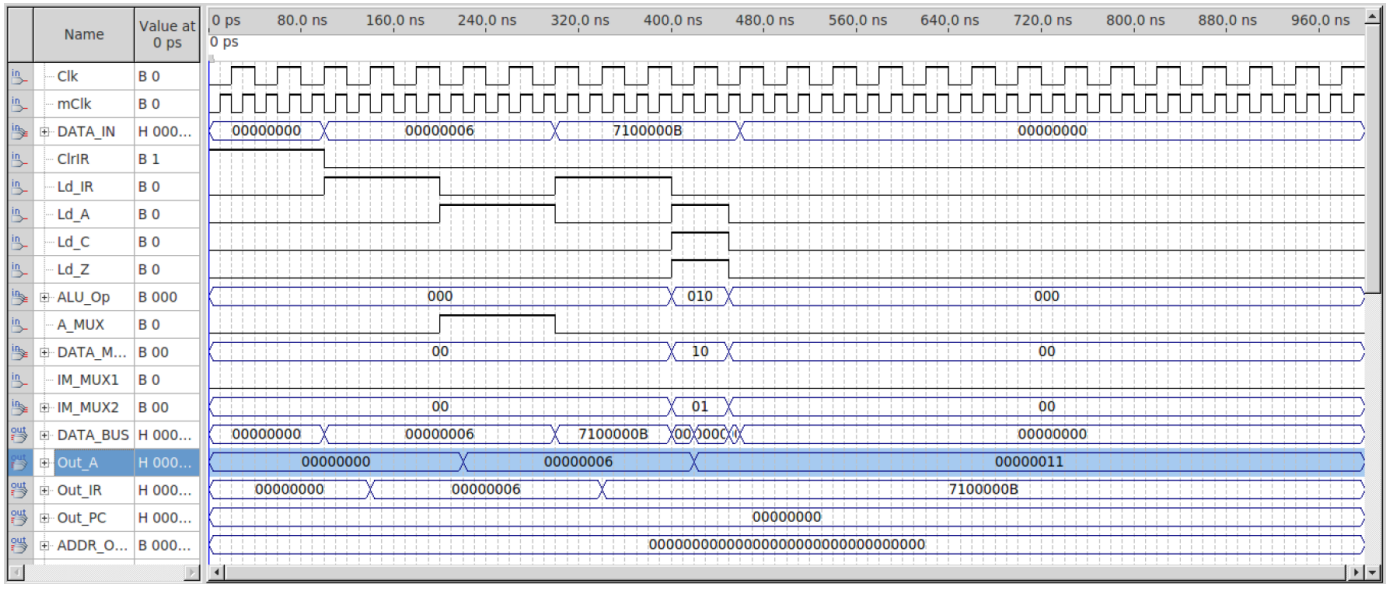
JMP


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA_IN
ClrIR
Ld_IR
Ld_PC
Out_IR
Out_PC
DATA_BUS
A_MUX
ADDR_O...
ALU_Op
B_MUX
Clr_A
Clr_B
Clr_C
Clr_Z
ClrPC
00000000
5000AAAA
0000AAAA
000

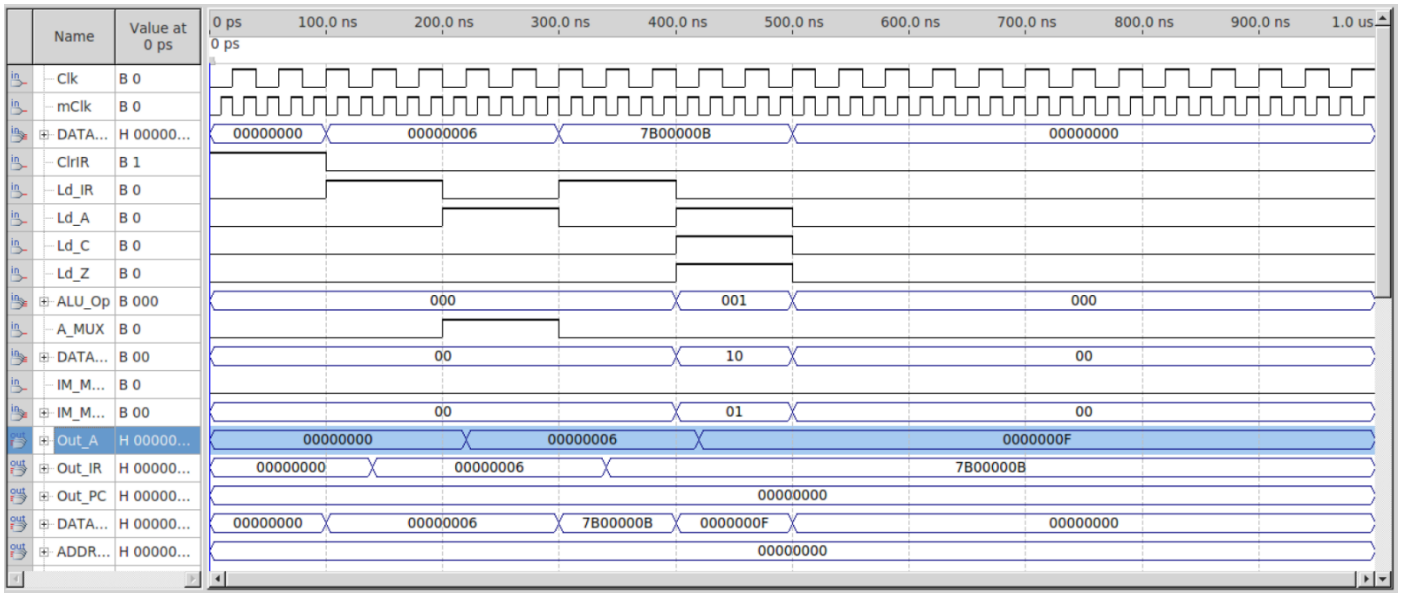
ANDI


Name
Value at 0 ps
0 ps
100,0 ns
200,0 ns
300,0 ns
400,0 ns
500,0 ns
600,0 ns
700,0 ns
800,0 ns
900,0 ns
1.0 us
Clk B 0
mClk B 0
DATA_IN H 0000...
00000000
00000006
7900000B
00000000
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Ld_C B 0
Ld_Z B 0
A_MUX B 0
DATA_... B 00
00
10
00
IM_MUX1 B 0
IM_MUX2 B 00
00
01
00
Out_A H 0000...
00000000
00000006
00000002
Out_IR H 0000...
00000000
00000006
7900000B
Out_PC H 0000...
00000000
DATA_... H 0000...
00000000
00000006
7900000B
00000002
00000000
ADDR_... B 0000...
00000000000000000000000000000000
ALU_Op B 000
000

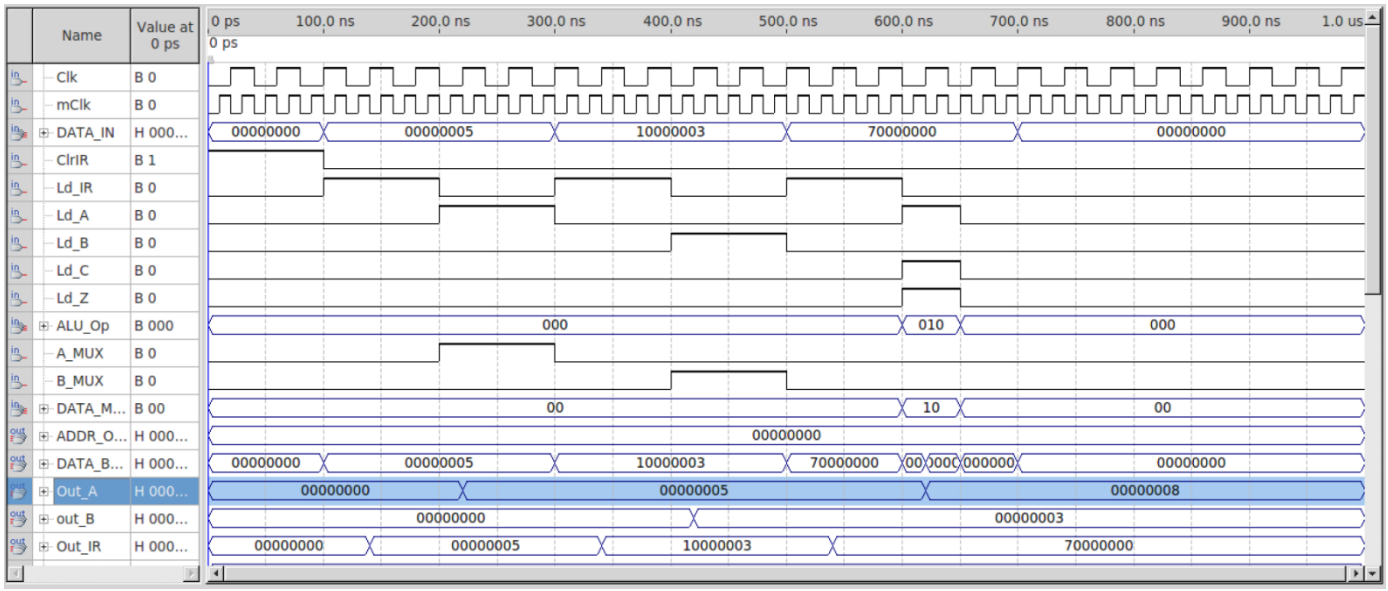
ADDI


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Clk B 0
mClk B 0
DATA_IN H 000...
00000000
00000006
7100000B
00000000
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
000
010
000
A_MUX B 0
DATA_M... B 00
00
10
00
IM_MUX1 B 0
IM_MUX2 B 00
00
01
00
DATA_BUS H 000...
00000000
00000006
7100000B
00000000
Out_A H 000...
00000000
00000006
00000011
Out_IR H 000...
00000000
00000006
7100000B
Out_PC H 000...
00000000
ADDR_O... B 000...
00000000000000000000000000000000

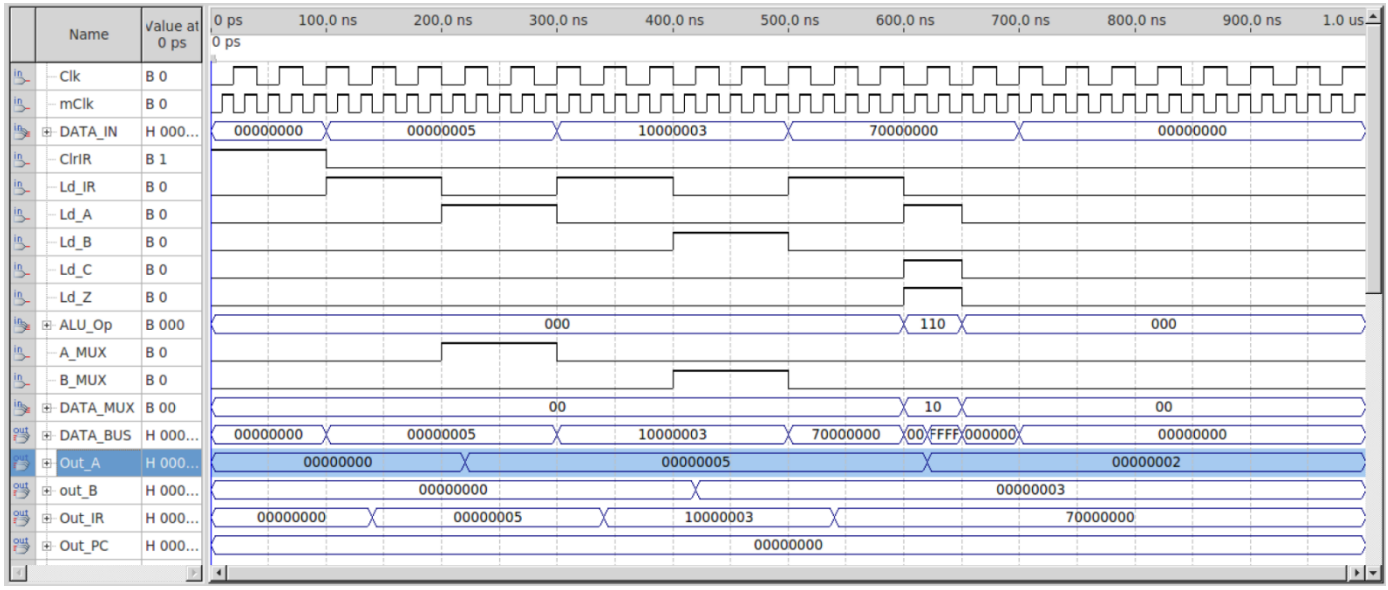
ORI


Name
Value at 0 ps
0 ps
100,0 ns
200,0 ns
300,0 ns
400,0 ns
500,0 ns
600,0 ns
700,0 ns
800,0 ns
900,0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
Ld_A
Ld_C
Ld_Z
ALU_Op
A_MUX
DATA...
IM_M...
IM_M...
Out_A
Out_IR
Out_PC
DATA...
ADDR...
00000000
00000006
7B00000B
00000000
000
001
000
00
10
00
00
01
00
0000000F
7B00000B
00000000

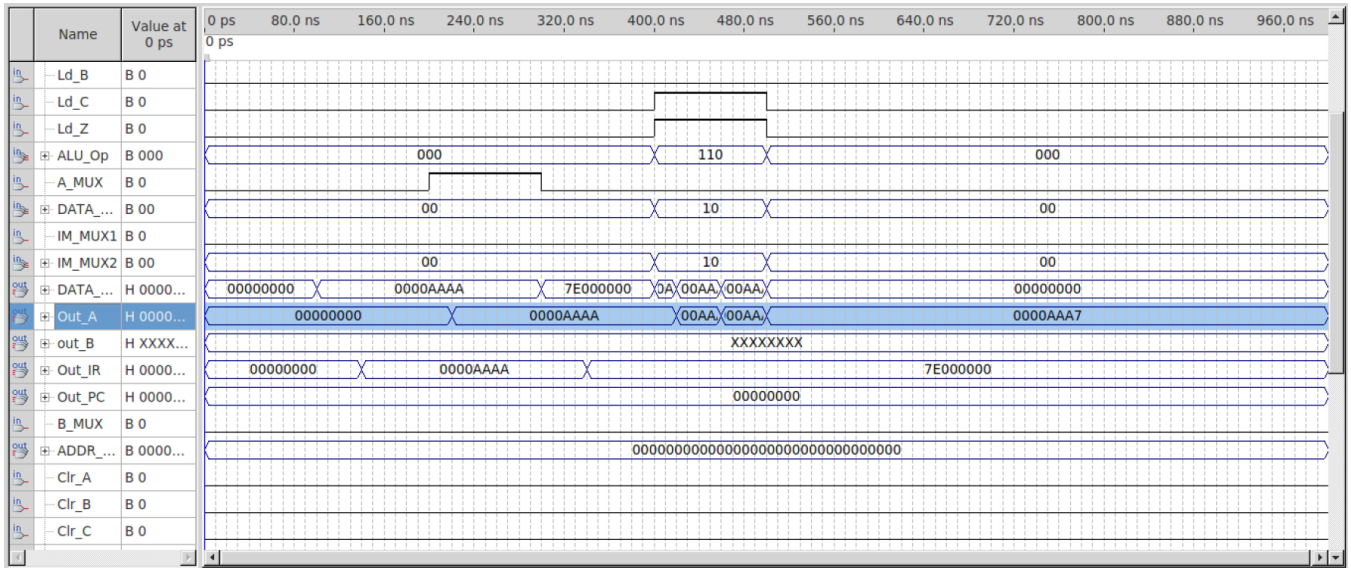
ADD


Name
Value at 0 ps
0 ps
100,0 ns
200,0 ns
300,0 ns
400,0 ns
500,0 ns
600,0 ns
700,0 ns
800,0 ns
900,0 ns
1.0 us
Clk
mClk
DATA_IN
ClrIR
Ld_IR
Ld_A
Ld_B
Ld_C
Ld_Z
ALU_Op
A_MUX
B_MUX
DATA_M...
ADDR_O...
DATA_B...
Out_A
out_B
Out_IR
00000000
00000005
10000003
70000000
00000008
000
010
00
10
00

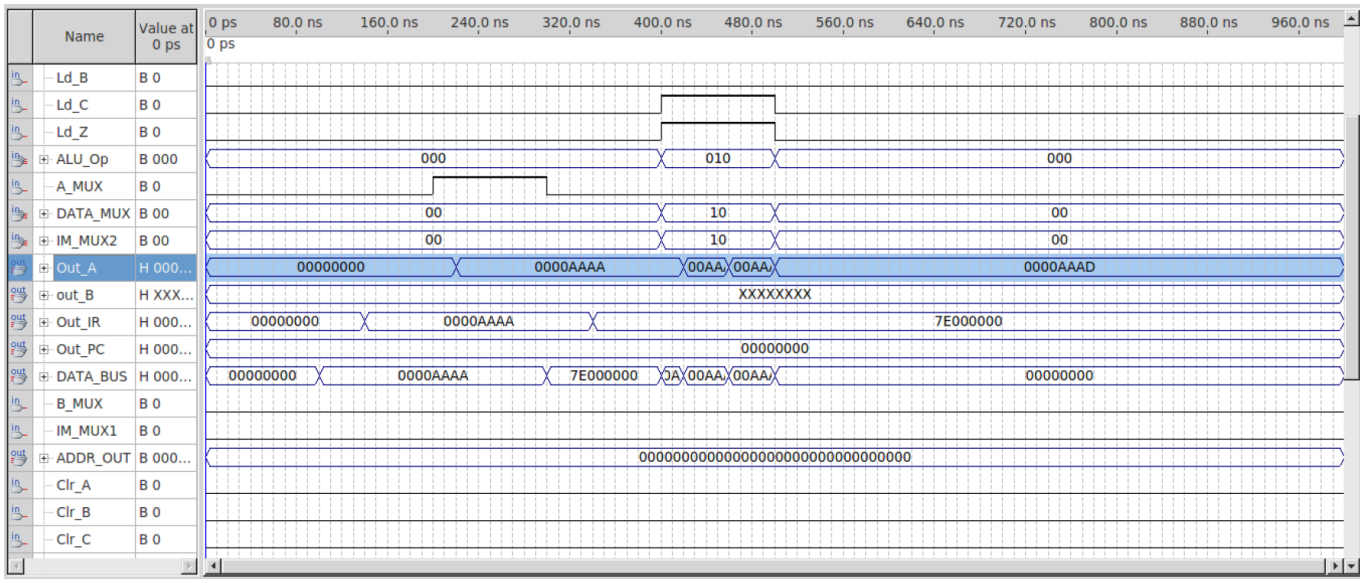
SUB


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk
mClk
DATA_IN
ClrIR
Ld_IR
Ld_A
Ld_B
Ld_C
Ld_Z
ALU_Op
A_MUX
B_MUX
DATA_MUX
DATA_BUS
Out_A
out_B
Out_IR
Out_PC
00000000
00000005
10000003
70000000
00000000
000
110
000
00
10
00
FFFF
00000002
00000003

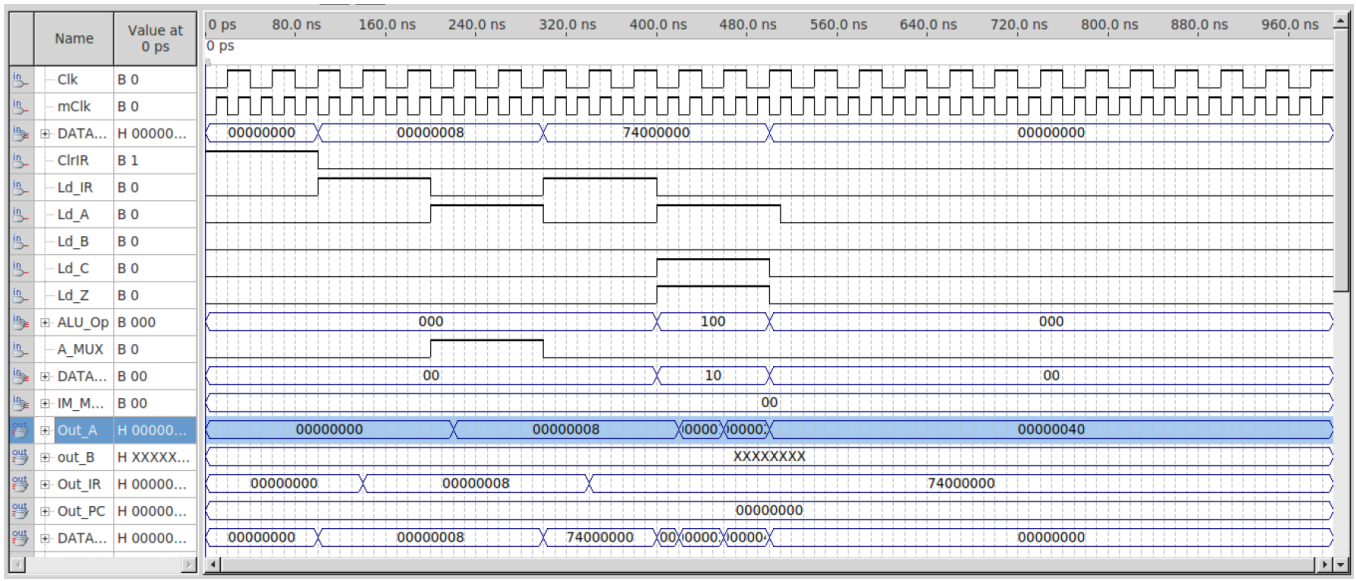
DECA


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Ld_B
Ld_C
Ld_Z
ALU_Op
A_MUX
DATA_...
IM_MUX1
IM_MUX2
DATA_...
Out_A
out_B
Out_IR
Out_PC
B_MUX
ADDR_...
Clr_A
Clr_B
Clr_C
000
110
000
00
10
00
00000000
0000AAAA
7E000000
00AA
00AA
0000AAA7
XXXXXXXX
00000000000000000000000000000000

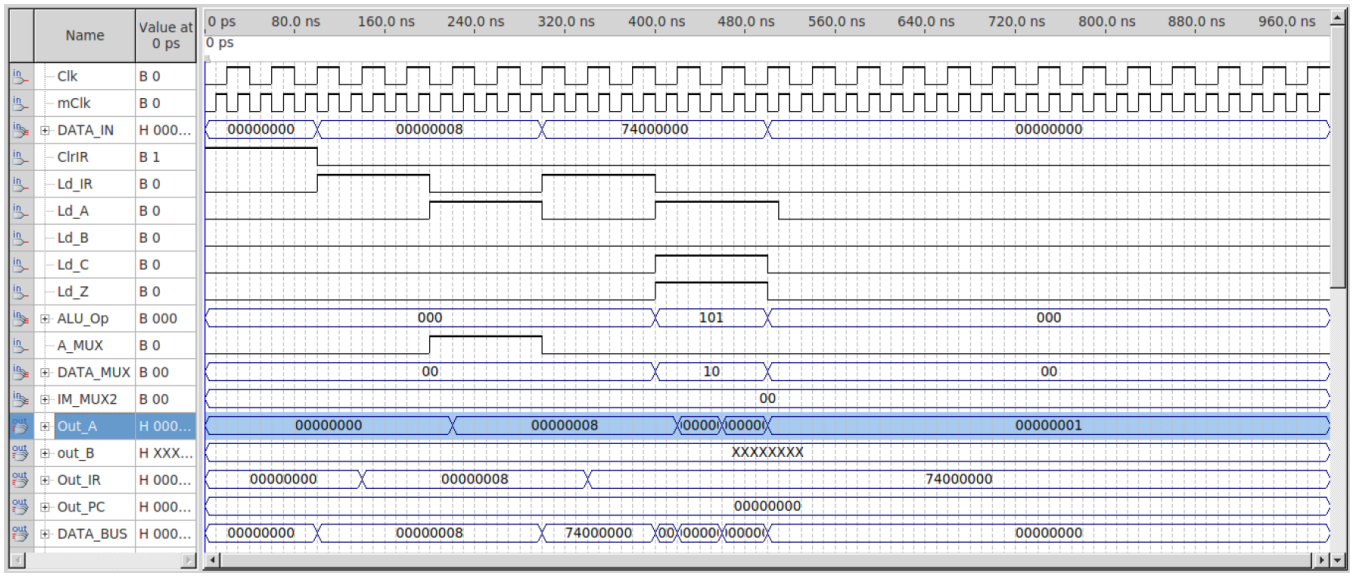
INCA


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Ld_B B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
000
010
000
A_MUX B 0
DATA_MUX B 00
00
10
00
IM_MUX2 B 00
00
10
00
Out_A H 000...
00000000
0000AAAA
00AA
00AA
0000AAAD
out_B H XXX...
XXXXXXXX
Out_IR H 000...
00000000
0000AAAA
7E000000
Out_PC H 000...
00000000
DATA_BUS H 000...
00000000
0000AAAA
7E000000
0A
00AA
00AA
00000000
B_MUX B 0
IM_MUX1 B 0
ADDR_OUT B 000...
00000000000000000000000000000000
Clr_A B 0
Clr_B B 0
Clr_C B 0

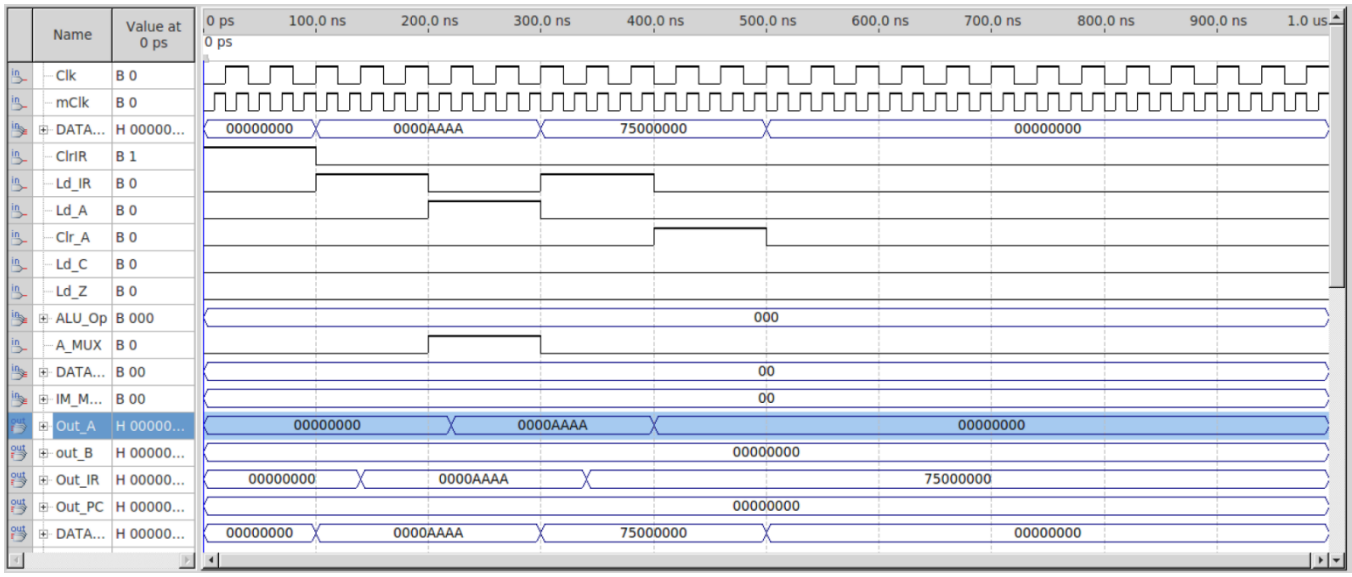
ROL


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Clk B 0
mClk B 0
DATA... H 00000...
00000000
00000008
74000000
00000000
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Ld_B B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
000
100
000
A_MUX B 0
DATA... B 00
00
10
00
IM_M... B 00
00
Out_A H 00000...
00000000
00000008
0000
0000
00000040
out_B H XXXXX...
XXXXXXXX
Out_IR H 00000...
00000000
00000008
74000000
Out_PC H 00000...
00000000
DATA... H 00000...
00000000
00000008
74000000
00
0000
0000
00000000

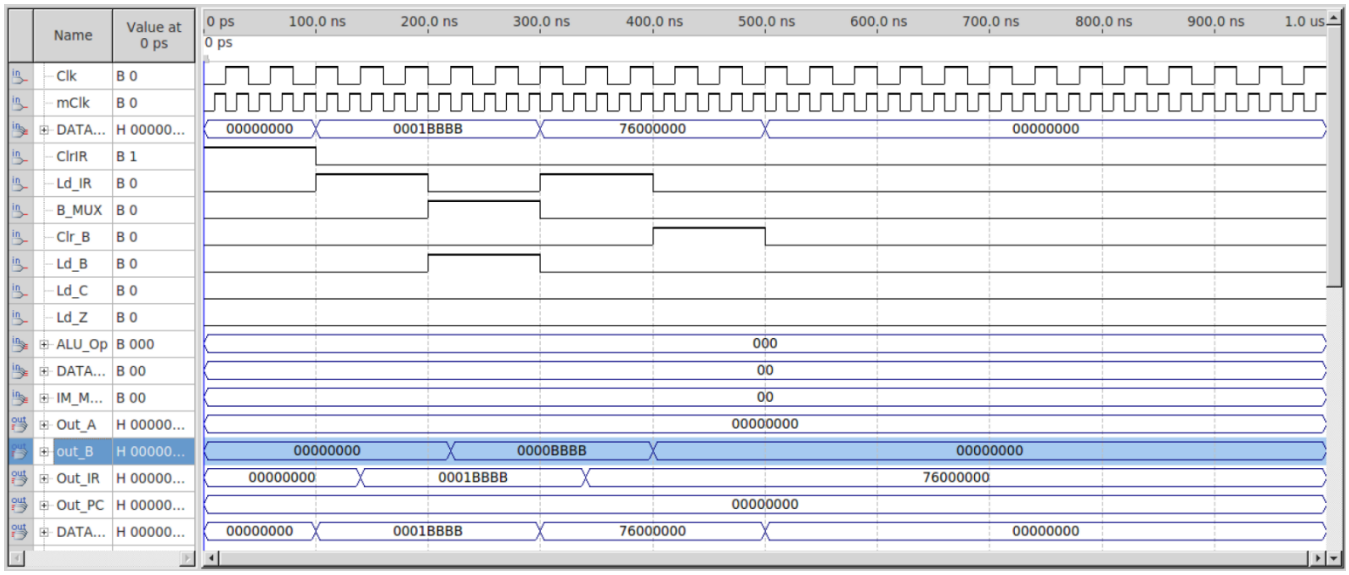
ROR


Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
Clk B 0
mClk B 0
DATA_IN H 000...
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Ld_B B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
A_MUX B 0
DATA_MUX B 00
IM_MUX2 B 00
Out_A H 000...
out_B H XXX...
Out_IR H 000...
Out_PC H 000...
DATA_BUS H 000...
00000000
00000008
74000000
00000000
000
101
000
00
10
00
00
00000000
00000008
0000
0000
00000001
XXXXXXXX
00000000
00000008
74000000
00000000
00000000
00000008
74000000
00
0000
0000
00000000

CLRA


Name
Value at 0 ps
0 ps
100.0 ns
200.0 ns
300.0 ns
400.0 ns
500.0 ns
600.0 ns
700.0 ns
800.0 ns
900.0 ns
1.0 us
Clk B 0
mClk B 0
DATA... H 00000...
ClrIR B 1
Ld_IR B 0
Ld_A B 0
Clr_A B 0
Ld_C B 0
Ld_Z B 0
ALU_Op B 000
A_MUX B 0
DATA... B 00
IM_M... B 00
Out_A H 00000...
out_B H 00000...
Out_IR H 00000...
Out_PC H 00000...
DATA... H 00000...
00000000
0000AAAA
75000000
00000000
000
00
00
00000000
0000AAAA
00000000
00000000
0000AAAA
75000000
00000000

CLRB


Name
Value at 0 ps
0 ps
100,0 ns
200,0 ns
300,0 ns
400,0 ns
500,0 ns
600,0 ns
700,0 ns
800,0 ns
900,0 ns
1.0 us
Clk
mClk
DATA...
ClrIR
Ld_IR
B_MUX
Clr_B
Ld_B
Ld_C
Ld_Z
ALU_Op
DATA...
IM_M...
Out_A
out_B
Out_IR
Out_PC
DATA...
00000000
0001BBBB
76000000
00000000
000
00
00
00000000
00000000
0000BBBB
00000000
00000000
0001BBBB
76000000
00000000
00000000
0001BBBB
76000000
00000000

INST	CLR_IR LD_IR	LD_PC INC_PC	CLR_A LD_A	CLR_B LD_B	CLR_C LD_C	CLR_Z LD_Z	ALU OP	EN WEN	A/B MUX	REG MUX	Data MUX	IM_MUX1 IM_MUX2
LDA	0/0	0/0	0/1	0/0	0/0	0/0	XXX	1/0	0/X	X	01	X
LDB	0/0	0/0	0/0	0/1	0/0	0/0	XXX	1/0	X/0	X	01	X
STA	0/0	0/0	0/0	0/0	0/0	0/0	XXX	1/1	X	0	X	X
STB	0/0	0/0	0/0	0/0	0/0	0/0	XXX	1/1	X	1	X	X
JMP	0/0	1/0	0/0	0/0	0/0	0/0	XXX	X	X	X	X	X
LDAI	0/0	0/0	0/1	0/0	0/0	0/0	XXX	X	1/X	X	X	X
LDBI	0/0	0/0	0/0	0/1	0/0	0/0	XXX	X	X/1	X	X	X
LUI	0/0	0/0	0/1	1/0	0/0	0/0	001	X	0/X	X	10	1/X
ANDI	0/0	0/0	0/1	0/0	0/1	0/1	0000	X	0/X	X	10	0/01
DECA	0/0	0/0	0/1	0/0	0/1	0/1	110	X	0/X	X	10	0/10
ADD	0/0	0/0	0/1	0/0	0/1	0/1	010	X	0/X	X	10	0/00
SUB	0/0	0/0	0/1	0/0	0/1	0/1	110	X	0/X	X	10	0/00
INCA	0/0	0/0	0/1	0/0	0/1	0/1	010	X	0/X	X	10	0/10
AND	0/0	0/0	0/1	0/0	0/1	0/1	000	X	0/X	X	10	0/00
ADDI	0/0	0/0	0/1	0/0	0/1	0/1	010	X	0/X	X	10	0/01
ORI	0/0	0/0	0/1	0/0	0/1	0/1	001	X	0/X	X	10	0/01
ROL	0/0	0/0	0/1	0/0	0/1	0/1	100	X	0/X	X	10	0/X
ROR	0/0	0/0	0/1	0/0	0/1	0/1	101	X	0/X	X	10	0/X
CLRA	0/0	0/0	1/0	0/0	0/0	0/0	XXX	X	X	X	X	X
CLRB	0/0	0/0	0/0	1/0	0/0	0/0	XXX	X	X	X	X	X
CLRC	0/0	0/0	0/0	0/0	1/0	0/0	XXX	X	X	X	X	X
CLRZ	0/0	0/0	0/0	0/0	0/0	1/0	XXX	X	X	X	X	X
PC <= PC+4	0/0	1/1	0/0	0/0	0/0	0/0	XXX	X	X	X	X	X
IR <= M[INST]	0/1	0/0	0/0	0/0	0/0	0/0	XXX	X	X	X	00	X
PC <= IR[15.0]	0/0	1/0	0/0	0/0	0/0	0/0	XXX	X	X	X	X	X

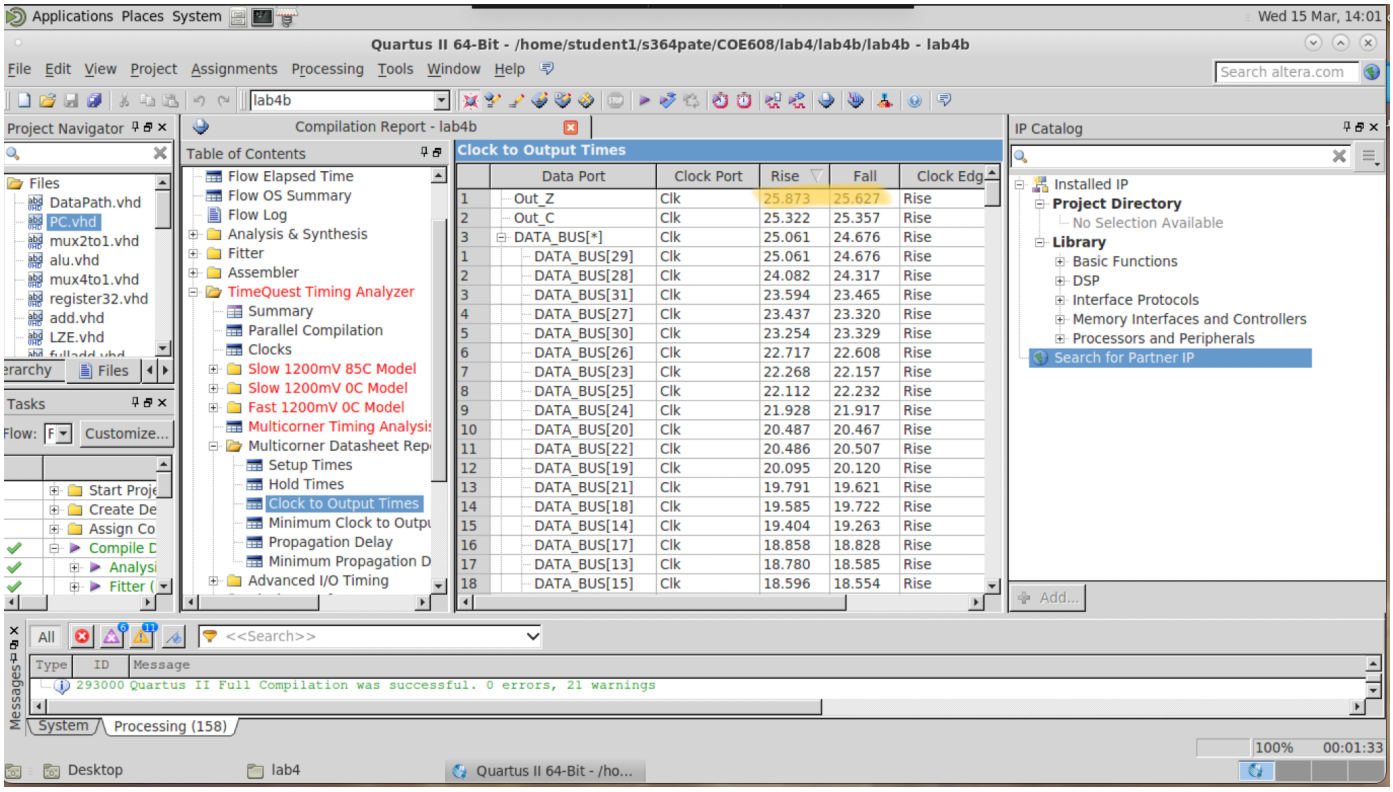
1. How does this data-path implement the INCA, ADDI, LDBI and LDA operations?

Each operation INCA, ADDI, LDBI, and LDA has specific requirements. INCA increments a register value by one using a register file and adder. ADDI adds an immediate value to a register value using a register file, adder, and a separate register to store the immediate value. LDBI loads an immediate value into a register using a register file and a separate register to store the immediate value. LDA loads a value from a memory address into a register using a memory unit and register file.

2. The data-path has a maximum reliable operating speed (Clk). What determines this speed? (i.e. how would you estimate the data-path circuit clock)?

To ensure that the data-path functions reliably, its highest operational speed is established by the critical path delay, which refers to the lengthiest path in the circuit. The delay is established by the slowest logic element in the path, encompassing the propagation delays of all gates and interconnects along the way. To estimate the circuit clock, identifying the critical path and determining the total delay along the path is crucial. It's important to set the clock frequency to a value that provides sufficient time for the propagation delay of the critical path to complete before the arrival of the next clock edge.

3. What is a reliable limit for your data-path clock?



(25.873ns+25.627ns)/2 = 25.75ns

Frequency = 1/25.75 = 0.0388GHz

Time = 0.0388 x 1000 = 38.83 ns

To determine a reliable limit for the data-path clock, the average duration of the most time-consuming rising and falling edges was calculated to be 25.75 nanoseconds, corresponding to a frequency of approximately 0.0388GHz. However, it should be considered that other factors may limit the maximum clock frequency. Therefore, a thorough analysis of the system is crucial to ensure that the chosen clock frequency is reliable.