Final project: Part V Memory subsystem and full integration

The machine language equivalent of the test program.

```
-- instruction number, Binary instruction, assembly Operation, Operations with values
0  => "00000000000000000000000000000000", -- nop                  ,
1  => "10000101111000000011111000000000", -- loadi rt, imm        , loadi r15, h01F0
2  => "11000100000011111111111110000000", -- brc ra, cond, off    , br r15,=0,-1
3  => "00101011111000000000000000011111", -- addi rt, ra, imm     , addi r31, r0, h001F
4  => "00100100001000000000000000000000", -- add rt, ra, rb       , add r1, r0, r0
5  => "00101100001000010000000000000000", -- inc rt, ra           , inc r1, r1
6  => "11000100000111110000001010000011", -- brc ra, cond, off    , br r31, <0, +5 (L2)
7  => "10101000000111110000000000000001", -- storr rb, ra         , storr r1, r31
8  => "00101100001000010000000000000000", -- inc rt, ra           , inc r1, r1
9  => "00001111111111110000000000000000", -- dec rt, ra           , dec r31, r31
10 => "11000000000000001111111000000000", -- jmp off              , jmp -4 (L1)
11 => "10000100010000000000001000000000", -- loadi rt, imm        , loadi r2, h0010
12 => "01010111110000000000000000000100", -- ori rt, ra, imm      , ori r30, r0, h0004
13 => "10001000011111100000000000000000", -- loadr rt, ra         , loadr r3, r30
14 => "10010000100111100000000001100000", -- loado rt, ra, off    , loado r4, r30, 3
15 => "01011011101000001111111111111111", -- xori rt, ra, imm     , xori r29, r0, hFFFF
16 => "10000000111000000000000000000000", -- move rt, ra          , move r7, r0
17 => "01010000101111010000000000010000", -- andi rt, ra, imm     , andi r5, r29, h0010
18 => "01010000110001000000000000000001", -- andi rt, ra, imm     , andi r6, r4, h0001
19 => "11000100000001100000000100000000", -- brc ra, cond, off    , br r6, =0, +2 (L4)
20 => "00100100111000110000000000000111", -- add rt, ra, rb       , add r7, r3, r7
21 => "01100100100001000000000000100000", -- shr rt, ra, n        , shr r4, r4, 1
22 => "01100000011000110000000000100000", -- shl rt, ra, n        , shl r3, r3, 1
23 => "00001100101001010000000000000000", -- dec rt, ra           , dec r5, r5
24 => "11000100000001011111110100000001", -- brc ra, cond, off    , br r5, ?0, -6 (L3)
25 => "10100100000000000000000000000111", -- stori rb, imm        , stori r7, 0
26 => "01101000111001110000000100100000", -- rol rt, ra, n        , rol r7, r7, 9
27 => "01101100111001110000000001100000", -- ror rt, ra, n        , ror r7, r7, 3
28 => "01000001000001110000000000000000", -- not rt, ra           , not r8, r7
29 => "01001101000111010000000000001000", -- xor rt, ra, rb       , xor r8, r29, r8
30 => "00000101001010000000000000000111", -- sub rt, ra, rb       , sub r9, r8, r7
31 => "00001001010010010000000000000001", -- subi rt, ra, imm     , subi r10, r9, h0001
32 => "11000100000010100000001001000100", -- brc ra, cond, off    , br r10, >0, +9 (Lx)
33 => "11000100000010100000001000000110", -- brc ra, cond, off    , br r10, ?0, +8 (Lx)
34 => "00101001011010100000000000000010", -- addi rt, ra, imm     , addi r11, r10, h0002
35 => "11000100000010110000001100000101", -- brc ra, cond, off    , br r11, ?0, +6 (Lx)
36 => "01001001100001110000000000001011", -- or rt, ra, rb        , or r12, r7, r11
37 => "10110000000000000000000001001010", -- storo rb, ra, off    , storo r12, r0, 1
38 => "01000101100010100000000000001011", -- and rt, ra, rb       , and r12, r12, r11
39 => "11000100000011000000000110000010", -- brc ra, cond, off    , br r12, =1, +3 (Lok)
40 => "11000000000000000000000000000000", -- jmp off              , jmp +0
41 => "11000000000000000000000000000000", -- jmp off              , jmp +0
42 => "10100100000000000111111000000111", -- stori rb, imm        , stori r7,h01F8
43 => "11000000000000000000000000000000", -- jmp off              , jmp +0
```

The commented VHDL code for the memory subsystem.

Top Entity :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- Top Entity for synchronous general purpose processor.
-- The top entity implements tested entitys to make up the system:
-- control unit for instruction register,sequencer and decode logic
-- proccessing unit name:data path,ALU and bank of 32 registers within
-- Memory Management unit (MMU)
-- Dual Port Memory 128x32  32 bit-word addressable
-- output reg 16 bits top 8MSBs output to LEDs on board
-- for more infomration regarding each entity see the corresponding file.

-- The system processes through the instructions as per the design supplied
-- The system should finish with the values:
--If the program has run correctly, execution should finish at
--instruction 43 and never move from there. Register and
--memory contents should be the following (decimal):
--r1 = 33; r2 = 16; r3 = r4 = r5 = r6 = r9 = 0;
--r7 = r8 = 44800; r10 = -1 (65535); r11 = r12 = 1;
--DMEM(0) = 700; DMEM(1) = 44801
--The LEDs should display decimal 44800.
entity Mem_Subsys_Full_Interg is
 GENERIC ( data_size : natural := 16;
           reg_size  : natural := 32);
    Port ( PB_Start  : in STD_LOGIC;
           clk       : in STD_LOGIC;
           PB_Rst    : in STD_LOGIC;
           LEDs      : out STD_LOGIC_VECTOR (7 downto 0));
end Mem_Subsys_Full_Interg;

architecture Behavioral of Mem_Subsys_Full_Interg is

-- internal RAM Instruction to Control unit
signal RAM_Inst_to_CTRL_Unit_int : STD_LOGIC_VECTOR(31 downto 0);

--signals to and from MMU
--data from ram to MMU internal signal
signal int_Data_Ram_In      : STD_LOGIC_VECTOR (31 downto 0);
--Instruction address from MMU to RAM internal signal
signal int_Inst_Add_Ram     : STD_LOGIC_VECTOR (6 downto 0);
-- data from RAM to MMU internal signal
signal int_Data_Ram_Out     : STD_LOGIC_VECTOR (31 downto 0);
-- data address internal signal
signal int_Data_Add         : STD_LOGIC_VECTOR (6 downto 0);
-- data from MMU to output reg internal signal
signal int_Output_Reg       : STD_LOGIC_VECTOR (15 downto 0);
-- write enable from MMU to RAM internal signal
signal int_Write_En_Ram     : STD_LOGIC;
-- write enable from MMU to Output register internal signal
signal int_Write_En_Out_Reg : STD_LOGIC;
```

```
--signals from control logic to Datapath
-- Register A to Datapath internal signal
signal int_RA_to_Datapath : STD_LOGIC_VECTOR(4 downto 0);
-- Register B to Datapath internal signal
signal int_RB_to_Datapath : STD_LOGIC_VECTOR(4 downto 0);
-- Write Address to Datapath internal signal
signal int_WA_to_Datapath : STD_LOGIC_VECTOR(4 downto 0);
-- Memory address to Datapath internal signal
signal int_MEM_ADDR_to_Datapath    : STD_LOGIC_VECTOR(data_size - 1 downto
0); --16 bit immidate value
-- Immediate value to Datapath internal signal
signal int_IMM_VALUE_to_Datapath   : STD_LOGIC_VECTOR(data_size - 1 downto
0); --16 bit immidate value
-- Address offset to Datapath internal signal
signal int_ADDR_OFFSET_to_Datapath : STD_LOGIC_VECTOR(9 downto 0);
-- Program offset to Datapath internal signal
signal int_PC_OFFSET_to_Datapath   : STD_LOGIC_VECTOR(8 downto 0);
-- Shift value to Datapath internal signal
signal int_SHIFTER_to_Datapath     : STD_LOGIC_VECTOR(3 downto 0);
-- branch CONDition to Datapath internal signal
signal int_COND_to_Datapath : STD_LOGIC_VECTOR(2 downto 0);
-- Output enable to Datapath internal signal
signal int_OEN_to_Datapath  : STD_LOGIC;
-- Multiplexer control selector Signal to Datapath internal signal
signal int_S_to_Datapath    : STD_LOGIC_VECTOR(2 downto 0);
-- Arithmatatic Logic Unit control Signal to Datapath internal signal
signal int_ALU_to_Datapath  : STD_LOGIC_VECTOR(3 downto 0);
-- Write enable to Datapath internal signal
signal int_WEN_to_Datapath  : STD_LOGIC;
-- Memory instruction address to datapath internal signal
signal int_MIA_to_Datapath  : STD_LOGIC_VECTOR(7 downto 0);
-- RAM data to datapath internal signal
signal int_RAM_DATA_to_Datapath       : STD_LOGIC_VECTOR ((data_size - 1)
downto 0);
--RAM DATA to data path internal signal
signal int_MEM_DATA_WRITE_to_Datapath : STD_LOGIC_VECTOR ((data_size - 1)
downto 0);
--RAM DATA ADDRESS internal signal
signal int_MEM_DATA_ADD_to_Datapath   : STD_LOGIC_VECTOR ((data_size - 1)
downto 0);
-- flags from ALU to the control unit internal signal
signal int_FLAGS_ALU_to_Datapath      : STD_LOGIC_VECTOR(7 downto 0);
-- LEDs value output internal signal
signal int_LEDs : STD_LOGIC_VECTOR(data_size-1 downto  0);



begin
-- output LEDS connected to top 8 MSBs
LEDs <= int_LEDs(15 downto 8);
```

```vhdl
Control_Logic: entity work.control_logic
PORT MAP   (--Inputs
            clk            => clk,
            rst            => PB_Rst,
            FLAGS_ALU      => int_FLAGS_ALU_to_Datapath,
            INSTRUCTION    => RAM_Inst_to_CTRL_Unit_int,
            --Outputs
            MIA            => int_MIA_to_Datapath,
            RA             => int_RA_to_Datapath,
            RB             => int_RB_to_Datapath,
            WA             => int_WA_to_Datapath,
            MEM_ADDR       => int_MEM_ADDR_to_Datapath,
            IMM_VALUE      => int_IMM_VALUE_to_Datapath,
            ADDR_OFFSET    => int_ADDR_OFFSET_to_Datapath,
            PC_OFFSET      => int_PC_OFFSET_to_Datapath,
            SHIFTER        => int_SHIFTER_to_Datapath,
            COND           => int_COND_to_Datapath,
            OEN            => int_OEN_to_Datapath,
            SEL            => int_S_to_Datapath,
            ALU            => int_ALU_to_Datapath,
            WEN            => int_WEN_to_Datapath);


DATAPATH: entity work.Param_Datapath
GENERIC MAP ( data_size => data_size,
              reg_size => reg_size)
PORT MAP(
         REG_A          => int_RA_to_Datapath,
         REG_B          => int_RB_to_Datapath,
         WRITE_EN       => int_WEN_to_Datapath,
         CLK            => clk,
         RST            => PB_Rst,
         WRITE_REG_ADDR => int_WA_to_Datapath,
         IMMED          => int_IMM_VALUE_to_Datapath,
         MEM_ADD        => int_MEM_ADDR_to_Datapath,
         Sel            => int_S_to_Datapath,
         ALU            => int_ALU_to_Datapath,
         SHIFT          => int_SHIFTER_to_Datapath,
         RAM_DATA       => int_RAM_DATA_to_Datapath,
         OEN            => int_OEN_to_Datapath,
         Flags          => int_FLAGS_ALU_to_Datapath,
         MEM_DATA_WRITE => int_MEM_DATA_WRITE_to_Datapath,
         MEM_DATA_ADD   => int_MEM_DATA_ADD_to_Datapath
         );

Output_reg: entity work.reg_bits
Generic map( data_size => data_size)
PORT MAP   ( --inputs
             clk       => clk,
             rst       => PB_rst,
             en        => int_Write_En_Out_Reg,
             DATA_IN   => int_Output_Reg,
             -- outputs
             DATA_OUT  => int_LEDs);
```

```
Dual_Port_RAM: entity work.Dual_Port_Mem
PORT MAP ( -- inputs
          clk          => clk,
          Inst_Add     => int_Inst_Add_Ram,
          Data_Add     => int_Data_Add,
          Write_Data_En=> int_Write_En_Ram,
          Data_In      => int_Data_Ram_Out,
          -- outputs
          Data_Out     => int_Data_Ram_In,
          Inst_Out     => RAM_Inst_to_CTRL_Unit_int);

Memory_Management_Unit: entity work.Mem_Manag_Unit
Port map (
          --inputs
          Start_PB         =>  PB_Start,
          Output_En_In     =>  int_OEN_to_Datapath,
          Inst_Add_In      =>  int_MIA_to_Datapath,
          Data_Proc_In     =>  int_MEM_DATA_WRITE_to_Datapath,
          Data_Add_In      =>  int_MEM_DATA_ADD_to_Datapath,
          Data_Ram_In      =>  int_Data_Ram_In,
          --outputs
          Inst_Add_Ram     =>  int_Inst_Add_Ram,
          Data_Ram_Out     =>  int_Data_Ram_Out,
          Data_Proc_Out    =>  int_RAM_DATA_to_Datapath,
          Data_Add         =>  int_Data_Add,
          Output_Reg       =>  int_Output_Reg,
          Write_En_Ram     =>  int_Write_En_Ram,
          Write_En_Out_Reg =>  int_Write_En_Out_Reg);

end Behavioral;
```

Control Logic in previous

Program counter in previous

Instruction register in previous

Datapath second level entity

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;
-- Architecture B Parameterizable Single Cycle Data Path
-- Synchronous read Register Bank connected to an Asynchronous
-- Data Path.
-- The Data path contains 3 multiplexers, a sinlge ALU and
-- a simulated ram Address output.
-- Multiplexer 1 controls the Input B of the ALU it either inputs
-- the immediate value or the value from Read B input from the register selected.
-- Multiplexer 2 controls what value is output to the RAM smulated address,
--it is either the output of the ALU or the input memory address.
--Multiplexer 3 controls the data that is input to the register bank input,
--either the output of the ALU is input or the value that is contained within
--address of the simulated RAM.
-- the simulated ram output is made up of a value to write to the ram
-- controlled by a tristate buffer an input for ram value to simulate the
-- value at that address and an output for the Address.
-- The ALU performs +1, -1, A+B, A-B, And, Or, Xor,Not operations,
--shift left & right, rotate left & right and the flags from
--the results are below
-- note only one of the bove ALU operations is able to be performed per clk cycle
-- Flags(0): OUT = 0
-- Flags(1): OUT ? 0
-- Flags(2): OUT = 1
-- Flags(3): OUT < 0
-- Flags(4): OUT > 0
-- Flags(5): OUT ? 0
-- Flags(6): OUT ? 0
-- Flags(7): Overflow

-- Sel mux selector - 3 bit
-- Sel(100) = Sel(MUX1 0 0);
-- Sel(010) = Sel(0 MUX2 0);
-- Sel(001) = Sel(0 0 MUX3);
entity Param_Datapath is
GENERIC ( data_size : natural := 16;
          reg_size  : natural := 32);
    Port ( REG_A          : in STD_LOGIC_VECTOR ((log2(reg_size))-1 downto 0);
           REG_B          : in STD_LOGIC_VECTOR ((log2(reg_size))-1 downto 0);
           WRITE_EN       : in STD_LOGIC;
        -- ENABLES REGISTERS TO BE WRITTEN TO WHEN VALUE 1
           CLK            : in STD_LOGIC;
           RST            : in STD_LOGIC;
           WRITE_REG_ADDR  : in STD_LOGIC_VECTOR ((log2(reg_size))- 1 downto 0);
        -- REGISTER NUMBER TO BE WRITTEN TO
           IMMED          : in STD_LOGIC_VECTOR ((data_size- 1) downto 0);
           MEM_ADD        : in STD_LOGIC_VECTOR ((data_size- 1) downto 0);
        -- MEMORY OF SIMULATED RAM TO BE CHOSEN
           Sel            : in STD_LOGIC_VECTOR (2 downto 0);
        -- BIT 0 CONTROLS mux 3, -- BIT 1 CONTROLS mux 2, -- BIT 2 CONTROLS mux 1
           ALU            : in STD_LOGIC_VECTOR (3 downto 0);
           SHIFT          : in STD_LOGIC_VECTOR ((log2(data_size))- 1 downto 0);
        -- number of shifts/rotations the ALU should infer on the data
           RAM_DATA       : in STD_LOGIC_VECTOR ((data_size - 1) downto 0);
        -- data from address of the RAM simulation
           OEN            : in STD_LOGIC;
        -- output enable to write to Ram simulation values.
           Flags          : out STD_LOGIC_VECTOR (7 downto 0);
           MEM_DATA_WRITE  : out STD_LOGIC_VECTOR ((data_size - 1) downto 0);
         --data to be written to the RAM simulation
           MEM_DATA_ADD    : out STD_LOGIC_VECTOR ((data_size - 1) downto 0));
        -- address of the RAM simulation to be read or written to
end Param_Datapath;
```

```
architecture Behavioral of Param_Datapath is

signal REG_A_DATA_INT        : STD_LOGIC_VECTOR((data_size-1) downto 0);
signal REG_B_DATA_INT        : STD_LOGIC_VECTOR((data_size-1) downto 0);
signal ALU_SHIFT_OUT_INT     : STD_LOGIC_VECTOR((data_size-1) downto 0);
signal MUX_1_OUT_INT         : STD_LOGIC_VECTOR((data_size-1) downto 0);
signal MUX_2_OUT_INT         : STD_LOGIC_VECTOR((data_size-1) downto 0);
signal MUX_3_OUT_INT         : STD_LOGIC_VECTOR((data_size-1) downto 0);
signal Flags_INT             : STD_LOGIC_VECTOR(7 downto 0);

begin

Reg_Bank: entity work.reg_bank
            Generic map( data_size => data_size,
                         reg_size  => reg_size)

              port map ( -- inputs
                         CLK        => clk ,
                         rst        => RST,
                         Write_en   => WRITE_EN,
                         Write_addr => WRITE_REG_ADDR,
                         Reg_A      => REG_A,
                         Reg_B      => REG_B,
                         DATA_IN    => MUX_3_OUT_INT,
                         -- outputs
                         DATA_OUT_1 => REG_A_DATA_INT,
                         DATA_OUT_2 => REG_B_DATA_INT);

ALU_Parameterizable: entity work.Parameterizable_ALU
            Generic map( data_size => data_size)

               Port map (-- inputs
                         A       => REG_A_DATA_INT,
                         B       => MUX_1_OUT_INT,
                         X       => SHIFT,
                         OpCode  => ALU,
                         -- outputs
                         ALU_Out => ALU_SHIFT_OUT_INT,
                         Flags   => Flags_INT);


  --MULTIPLEXER 1
 MUX_1_OUT_INT   <= IMMED when Sel(2) = '1' else--
                    REG_B_DATA_INT when Sel(2) = '0' else --
                    (others => 'U');
  --MULTIPLEXER 2
 MUX_2_OUT_INT   <=  MEM_ADD when Sel(1) = '1' else--
                     ALU_SHIFT_OUT_INT when Sel(1) = '0' else--
                     (others => 'U');
   --MULTIPLEXER 3
 MUX_3_OUT_INT   <= RAM_DATA when Sel(0) = '1' else--
                    ALU_SHIFT_OUT_INT when Sel(0) = '0' else--
                    (others => 'U');

 MEM_DATA_WRITE <= REG_B_DATA_INT;
 MEM_DATA_ADD   <= MUX_2_OUT_INT;
 Flags <= Flags_INT;

end Behavioral;
```

Reg bank inside of datapath already exists

ALU_parameterizable inside of datapath exists in previous

Output reg third level entity

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Parametrizable D type Flip flop with enable
-- recevies generic data_size

entity reg_bits is
    GENERIC( data_size : natural := 16);
    Port (
           -- Inputs
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           en : in STD_LOGIC;
           -- Outputs
           DATA_IN : in STD_LOGIC_VECTOR (data_size -1 downto 0);
           DATA_OUT : out STD_LOGIC_VECTOR (data_size -1 downto 0));
end reg_bits;

architecture Behavioral of reg_bits is

begin
process(clk)
   begin
     if(rising_edge(clk)) then
         if(rst = '1') then
             DATA_OUT <=(others => '0');
         elsif(en = '1') then
             DATA_OUT <= DATA_IN;
         End if;
     End if;
   end process;

end Behavioral;
```

Dual port Ram third level entity

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- This Dual Port Memory is a 128 address 0-127, 32 bit data ram.
-- The 1st 64 addresses (0-63) contain instructions and the final
-- 64-127 will be data.
-- The dual reads are Asynchronous
-- The single writes are synchronous
-- NO RESET avaliable
-- read commands output a 32 bit word for both instructions and data
-- input data is always a 32 bit word
-- all addresses input to the memory are 7 bits
entity Dual_Port_Mem is
    Port (
           clk           : in STD_LOGIC;
           Inst_Add      : in STD_LOGIC_VECTOR (6 downto 0);
           Data_Add      : in STD_LOGIC_VECTOR (6 downto 0);
           Write_Data_En : in STD_LOGIC;
           Data_In       : in STD_LOGIC_VECTOR (31 downto 0);
           Data_Out      : out STD_LOGIC_VECTOR (31 downto 0);
           Inst_Out      : out STD_LOGIC_VECTOR (31 downto 0));


end Dual_Port_Mem;


architecture Behavioral of Dual_Port_Mem is

type ram_type is array (0 to 127)of std_logic_vector (31 downto 0);
signal my_ram: ram_type := (
-- instruction memory
0  => "00000000000000000000000000000000", -- nop                 ,
1  => "10000101111000000011111000000000", -- loadi rt, imm       , loadi r15, h01F0
2  => "11000100000011111111111110000000", -- brc ra, cond, off   , br r15,=0,-1
3  => "00101011111000000000000000011111", -- addi rt, ra, imm    , addi r31, r0,
h001F
4  => "00100100001000000000000000000000", -- add rt, ra, rb      , add r1, r0, r0
5  => "00101100001000010000000000000000", -- inc rt, ra          , inc r1, r1
6  => "11000100000111110000001010000011", -- brc ra, cond, off   , br r31, <0, +5
(L2)
7  => "10101000000111110000000000000001", -- storr rb, ra        , storr r1, r31
8  => "00101100001000010000000000000000", -- inc rt, ra          , inc r1, r1
9  => "00001111111111110000000000000000", -- dec rt, ra          , dec r31, r31
10 => "11000000000000000111111000000000", -- jmp off             , jmp -4 (L1)
11 => "10000100010000000000001000000000", -- loadi rt, imm       , loadi r2, h0010
12 => "01010111110000000000000000000100", -- ori rt, ra, imm     , ori r30, r0,
h0004
13 => "10001000011111100000000000000000", -- loadr rt, ra        , loadr r3, r30
14 => "10010000100111100000000001100000", -- loado rt, ra, off   , loado r4, r30, 3
15 => "01011011101000001111111111111111", -- xori rt, ra, imm    , xori r29, r0,
hFFFF
16 => "10000000111000000000000000000000", -- move rt, ra         , move r7, r0
17 => "01010000101111010000000000010000", -- andi rt, ra, imm    , andi r5, r29,
h0010
18 => "01010000110001000000000000000001", -- andi rt, ra, imm    , andi r6, r4,
h0001
```

```vhdl
19 => "11000100000001100000000100000000", -- brc ra, cond, off     , br r6, =0, +2 (L4)
20 => "00100100111000110000000000000111", -- add rt, ra, rb        , add r7, r3, r7
21 => "01100100100001000000000000100000", -- shr rt, ra, n         , shr r4, r4, 1
22 => "01100000011000110000000000100000", -- shl rt, ra, n         , shl r3, r3, 1
23 => "00001100101001010000000000000000", -- dec rt, ra            , dec r5, r5
24 => "11000100000001011111110100000001", -- brc ra, cond, off     , br r5, ?0, -6 (L3)
25 => "10100100000000000000000000000111", -- stori rb, imm         , stori r7, 0
26 => "01101000111001110000000100100000", -- rol rt, ra, n         , rol r7, r7, 9
27 => "01101100111001110000000001100000", -- ror rt, ra, n         , ror r7, r7, 3
28 => "01000001000001110000000000000000", -- not rt, ra            , not r8, r7
29 => "01001101000111010000000000001000", -- xor rt, ra, rb        , xor r8, r29, r8
30 => "00000101001010000000000000000111", -- sub rt, ra, rb        , sub r9, r8, r7
31 => "00001001010010010000000000000001", -- subi rt, ra, imm      , subi r10, r9, h0001
32 => "11000100000010100000010010000100", -- brc ra, cond, off     , br r10, >0, +9 (Lx)
33 => "11000100000010100000010000000110", -- brc ra, cond, off     , br r10, ?0, +8 (Lx)
34 => "00101001011010100000000000000010", -- addi rt, ra, imm      , addi r11, r10, h0002
35 => "11000100000010110000001100000101", -- brc ra, cond, off     , br r11, ?0, +6 (Lx)
36 => "01001001100001110000000000001011", -- or rt, ra, rb         , or r12, r7, r11
37 => "10110000000000000000000001001010", -- storo rb, ra, off     , storo r12, r0, 1
38 => "01000101100010100000000000001011", -- and rt, ra, rb        , and r12, r12, r11
39 => "11000100000011000000001100000010", -- brc ra, cond, off     , br r12, =1, +3 (Lok)
40 => "11000000000000000000000000000000", -- jmp off               , jmp +0
41 => "11000000000000000000000000000000", -- jmp off               , jmp +0
42 => "10100100000000000011111100000111", -- stori rb, imm         , stori r7,h01F8
43 => "11000000000000000000000000000000", -- jmp off               , jmp +0
-- all other addresses are data memory
others => X"00000000");

begin
-- instruction Asynchronous read
Inst_Out <= my_ram(to_integer(unsigned(Inst_Add)));
-- data Asynchronous read
Data_Out <= my_ram(to_integer(unsigned(Data_Add)));

-- Synchronous Write, NO RESET
process (clk)
begin
    if(rising_edge(clk)) then
        if (Write_Data_En = '1') then
        my_ram(to_integer(unsigned(Data_Add))) <= Data_In;
        end if;
    end if;
end process;
end Behavioral;
```

Memory management unit third level entity

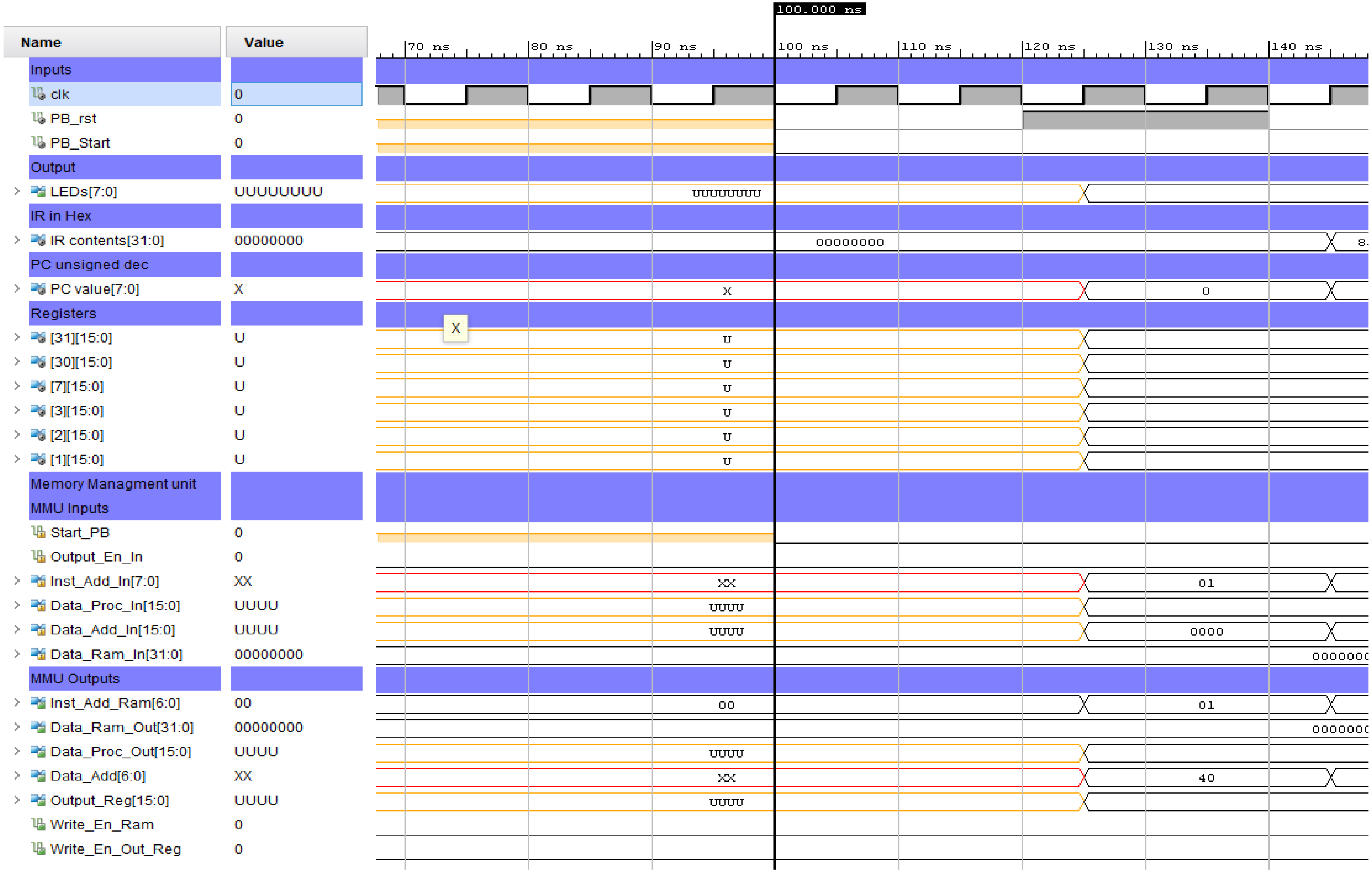
```vhdl
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-- Asynchronous combinational Memory Management unit (MMU)
-- address range = X'0100-X'01ff for peripherals
-- start push button value is at address X'01F0
-- LED's address X'01F8
-- 8 LEDs on board so 8 MSBs of the 16 bit data sent to output reg (LEDs)
-- ram data/instruction accesses kept separate within the MMU
-- no page management implemented
-- data 16 bits from processor
-- data 32 bits from RAM, made up of 2 16 bit sections of data
-- MMU makes 128 virtual addresses in the 64 data addresses in ram


entity Mem_Manag_Unit is
    Port (
           --PB = push button
           --En = enable
           --Add = address
           --Proc = processing
           --Reg = register
           --inputs
           Start_PB         : in STD_LOGIC;
           Output_En_In     : in STD_LOGIC;
           Inst_Add_In      : in STD_LOGIC_VECTOR (7 downto 0);
           Data_Proc_In     : in STD_LOGIC_VECTOR (15 downto 0);
           Data_Add_In      : in STD_LOGIC_VECTOR (15 downto 0);
           Data_Ram_In      : in STD_LOGIC_VECTOR (31 downto 0);
           --outputs
           Inst_Add_Ram     : out STD_LOGIC_VECTOR (6 downto 0);
           Data_Ram_Out     : out STD_LOGIC_VECTOR (31 downto 0);
           Data_Proc_Out    : out STD_LOGIC_VECTOR (15 downto 0);
           Data_Add         : out STD_LOGIC_VECTOR (6 downto 0);
           Output_Reg       : out STD_LOGIC_VECTOR (15 downto 0);
           Write_En_Ram     : out STD_LOGIC;
           Write_En_Out_Reg : out STD_LOGIC);

end Mem_Manag_Unit;

architecture Behavioral of Mem_Manag_Unit is

  -- start push button 1 bit value resized to 16 bits
  signal Start_PB_resize : std_logic_vector(15 downto 0);
  -- data written to memory 16 MSB or LSB has been overwritten
  signal edited_memory_data: std_logic_vector(31 downto 0);
  -- LSBs of Data from RAM memory address
  signal int_LSB_16bits  : std_logic_vector(15 downto 0);
  -- MSBs of Data from RAM memory address
  signal int_MSB_16bits  : std_logic_vector(15 downto 0);

begin

 -- Instruction address selection for RAM when MSB = 0
 Inst_Add_Ram <= Inst_Add_In(6 downto 0) when Inst_Add_In(7) = '0' else
              (others => '0');

 -- physical data address +64 offset for data section in RAM
 Data_Add     <= std_logic_vector(unsigned(Data_Add_In(7 downto 1))+ 64);

 -- output reg address write enable signal functionality
 Write_En_Out_Reg <= '1' when ((Output_En_In = '1') and (Data_Add_In = X"01F8"))else
'0';
```

```vhdl
-- output reg data line connection to MMU data bus
 Output_Reg <= Data_Proc_In;

 -- 1 bit push button value resized to be 16 bits, converted to data
 Start_PB_resize<= X"0001" when Start_PB ='1' else
                   (others=> '0');

-- ram write enable signal connections
   Write_En_Ram <= '1' when (Output_En_In = '1') else
                 '0';

 -- Data from the MMU to the Processor
 -- push button 16 bit value of X'0001 written to processor when address is X"01F0"
 -- when data address in bit (0) value is 0 the 16 LSBs from rams 32 bit data are written to processor
 -- when data address in bit (0) value is 1 the 16 MSBs from rams 32 bit data are written to processor
 Data_Proc_Out <= Start_PB_resize when (Data_Add_In = X"01F0") else
                  Data_Ram_In (15 downto 0) when Data_Add_In(0) = '0' else
                  Data_Ram_In (31 downto 16)when Data_Add_In(0) = '1' else
                  (others => 'U');

--The 16 LSB from  RAMs 32 bit memory data written to internal signal
 int_LSB_16bits <= Data_Ram_In(15 downto 0);
 --The 16 MSB from  RAMs 32 bit memory data written to internal signal
 int_MSB_16bits <= Data_Ram_In(31 downto 16);

 -- data to RAM address organised via data address bit (0) from processor,
 -- if address bit value (0) is 0 then the 16LSBs are over written keeping
 -- the original 16 MSBs.
 -- if address bit value (0) is 1 then the 16MSBs are over written keeping
 -- the original 16 LSBs.

 edited_memory_data(31 downto 0) <= Data_Proc_In & int_LSB_16bits when Data_Add_In(0) = '1' else
                                    int_MSB_16bits & Data_Proc_In when Data_Add_In(0) = '0' else
                                    (others => '0');

 -- updated 32 bit RAM data for address selected written back to RAM
 Data_Ram_Out<= edited_memory_data;

 end Behavioral;
```

The commented VHDL testbench:

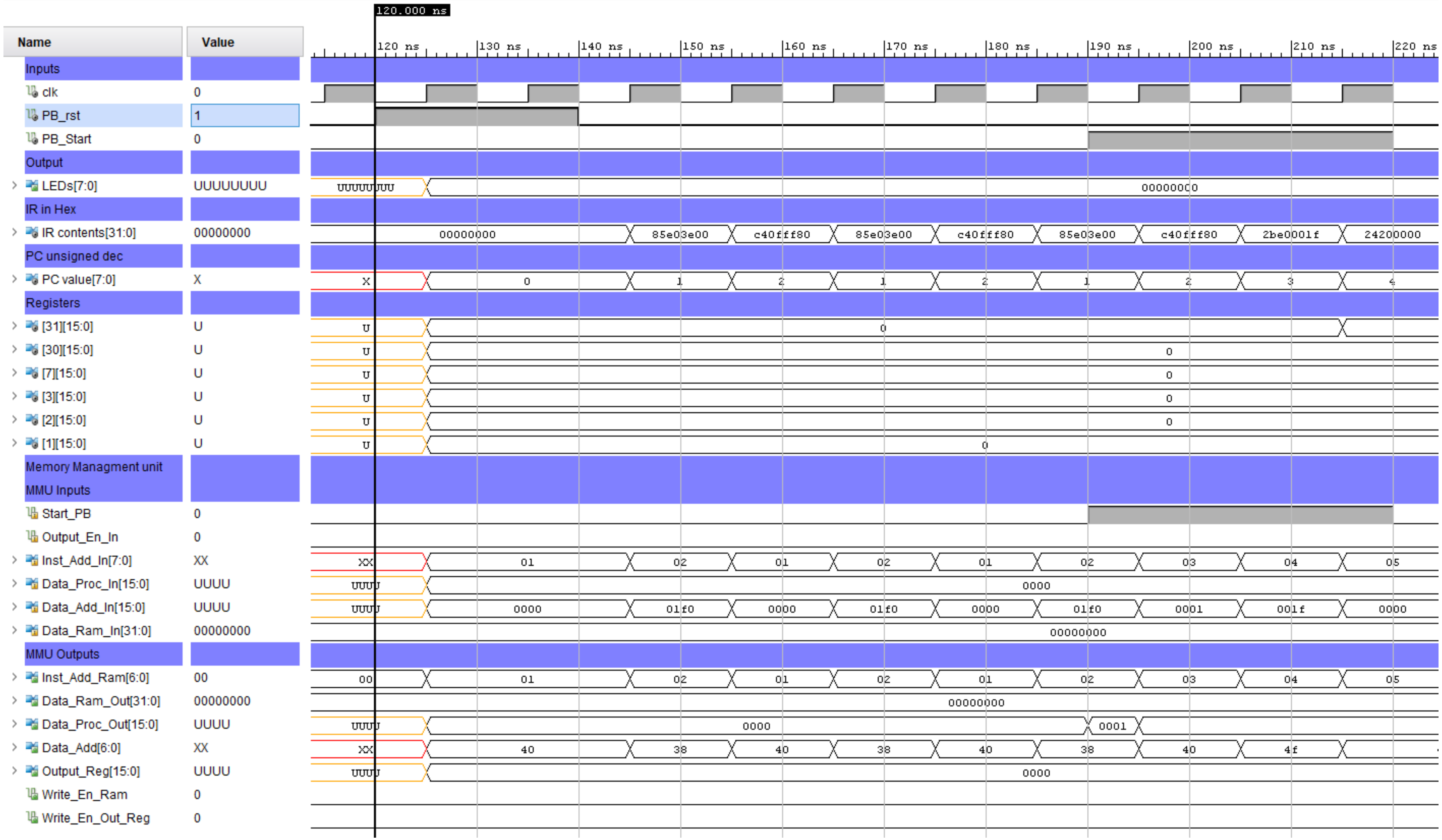
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

--Testing strategy is reset the entire system to known values,
-- check that the program counter does not exceed 2 in simulations until
the
-- push button has been pressed and then allow the code runs through the
-- sequence and check simulation results are correct until the final
-- output reg value is dec value 44800.

entity Mem_Subsys_Full_TB is

end Mem_Subsys_Full_TB;

architecture Behavioral of Mem_Subsys_Full_TB is

constant clk_period : time := 10ns;
--inputs
signal clk : STD_LOGIC;
signal PB_rst : STD_LOGIC;
signal PB_Start: STD_LOGIC;

begin

UUT: entity work.Mem_Subsys_Full_Interg
PORT MAP(
         clk => clk,
         PB_Start=> PB_Start,
         PB_Rst  => PB_rst);

-- Clock process
 clk_process :process
 begin
 clk <= '0';
 wait for clk_period/2;
 clk <= '1';
 wait for clk_period/2;
 end process;

-- Test procedure
TEST: process
begin
wait for 100 ns; --wait for initialization
wait until falling_edge(CLK);-- input signals change at falling edge
-- reset registers
PB_rst <= '0';
PB_Start  <= '0';
wait for clk_period*2;
PB_rst <= '1';
wait for clk_period*2;
PB_rst <= '0';
wait for clk_period*5;
PB_Start  <= '1';
wait for clk_period*3;
PB_Start  <= '0';

wait;
end process;

end Behavioral;
```

Screenshots of the simulation

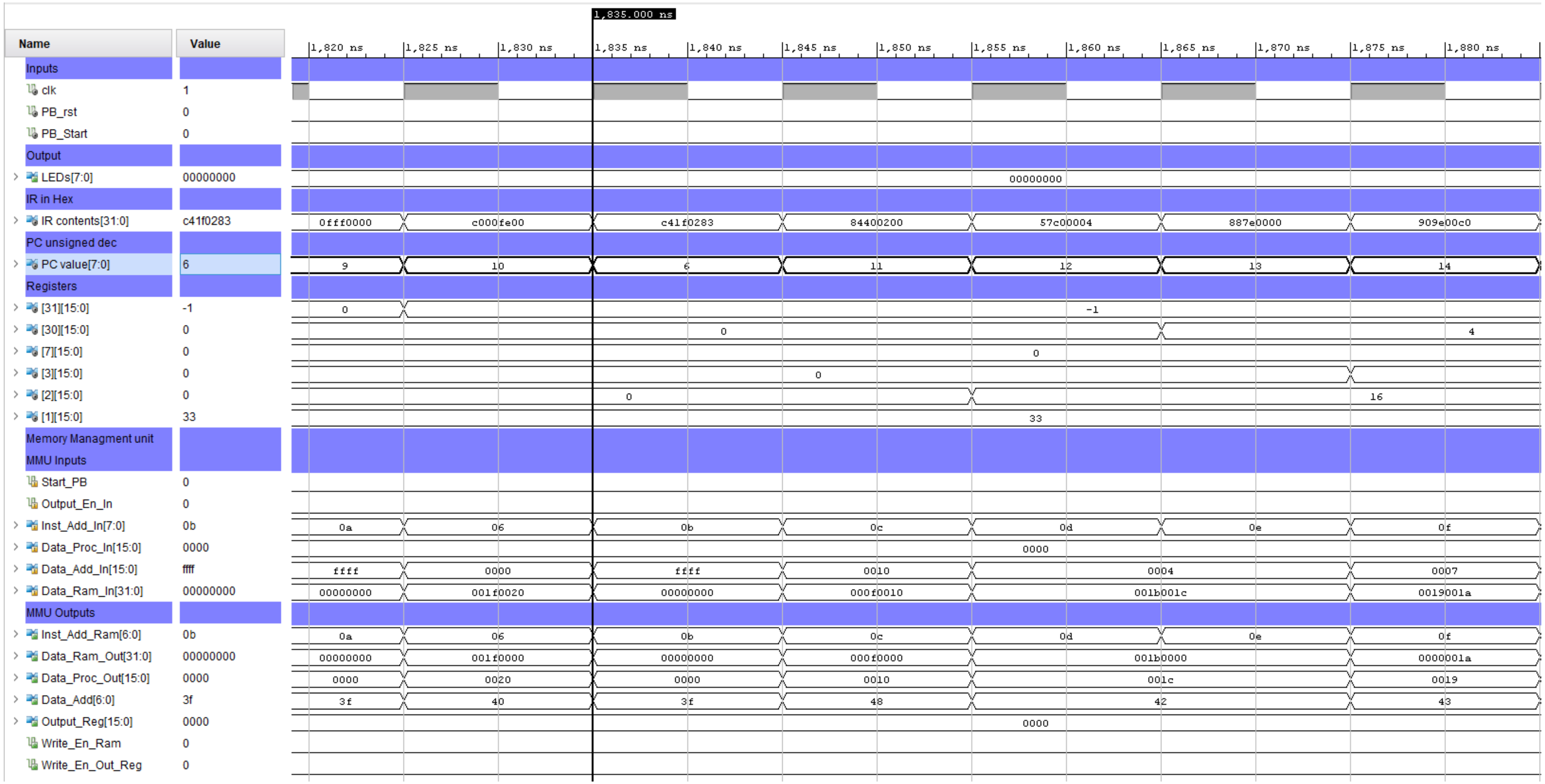


Name	Value
Inputs	
clk	0
PB_rst	0
PB_Start	0
Output	
LEDs[7:0]	UUUUUUUU
IR in Hex	
IR contents[31:0]	00000000
PC unsigned dec	
PC value[7:0]	X
Registers	
[31][15:0]	U
[30][15:0]	U
[7][15:0]	U
[3][15:0]	U
[2][15:0]	U
[1][15:0]	U
Memory Managment unit	
MMU Inputs	
Start_PB	0
Output_En_In	0
Inst_Add_In[7:0]	XX
Data_Proc_In[15:0]	UUUU
Data_Add_In[15:0]	UUUU
Data_Ram_In[31:0]	00000000
MMU Outputs	
Inst_Add_Ram[6:0]	00
Data_Ram_Out[31:0]	00000000
Data_Proc_Out[15:0]	UUUU
Data_Add[6:0]	XX
Output_Reg[15:0]	UUUU
Write_En_Ram	0
Write_En_Out_Reg	0

100ns initialisation



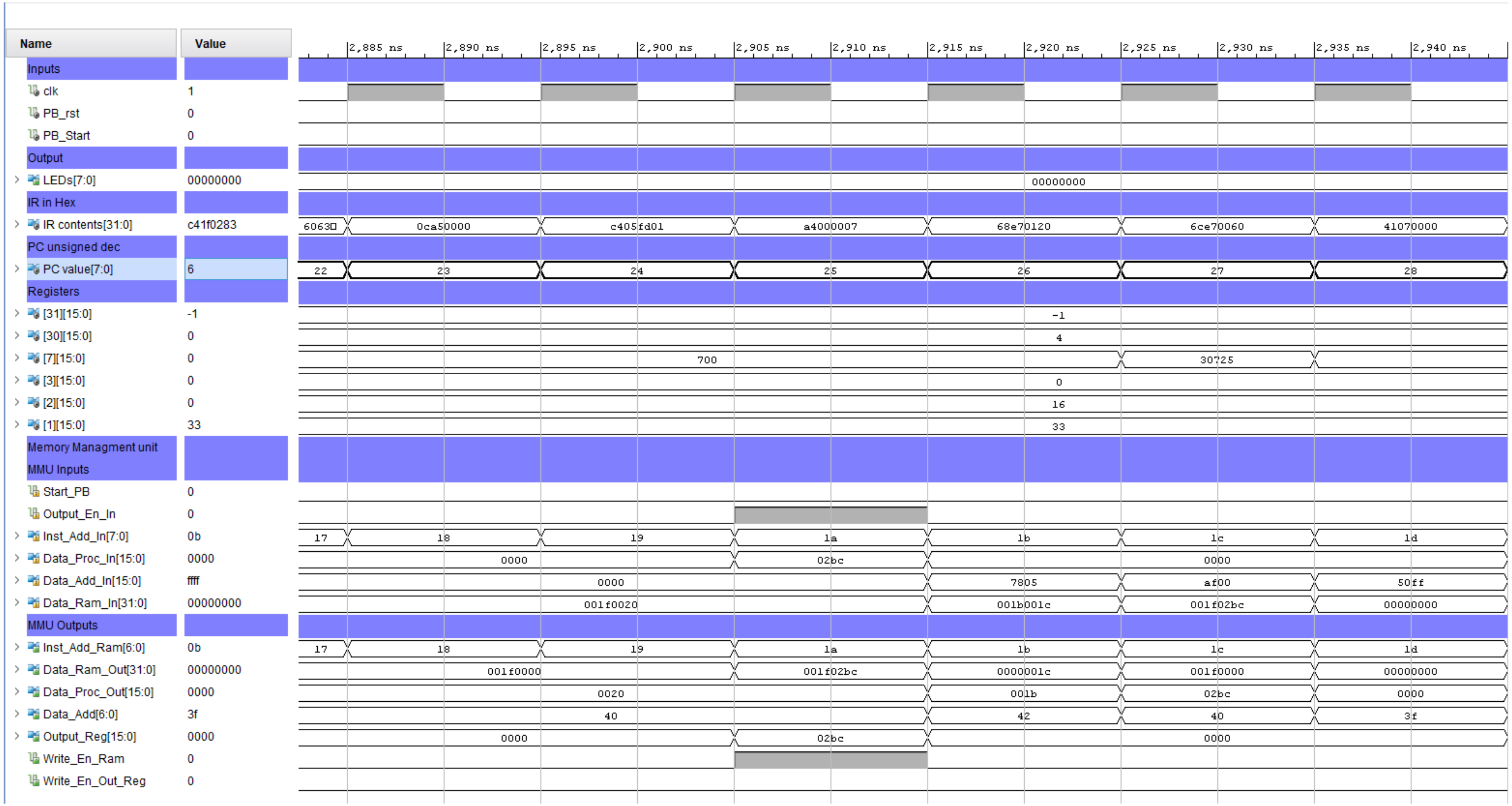
Push button reset allowing program to Start & Pb start allowing PC value to increment above 2



Name	Value
Inputs	
clk	1
PB_rst	0
PB_Start	0
Output	
LEDs[7:0]	00000000
IR in Hex	
IR contents[31:0]	c41f0283
PC unsigned dec	
PC value[7:0]	6
Registers	
[31][15:0]	-1
[30][15:0]	0
[7][15:0]	0
[3][15:0]	0
[2][15:0]	0
[1][15:0]	33
Memory Managment unit	
MMU Inputs	
Start_PB	0
Output_En_In	0
Inst_Add_In[7:0]	0b
Data_Proc_In[15:0]	0000
Data_Add_In[15:0]	ffff
Data_Ram_In[31:0]	00000000
MMU Outputs	
Inst_Add_Ram[6:0]	0b
Data_Ram_Out[31:0]	00000000
Data_Proc_Out[15:0]	0000
Data_Add[6:0]	3f
Output_Reg[15:0]	0000
Write_En_Ram	0
Write_En_Out_Reg	0

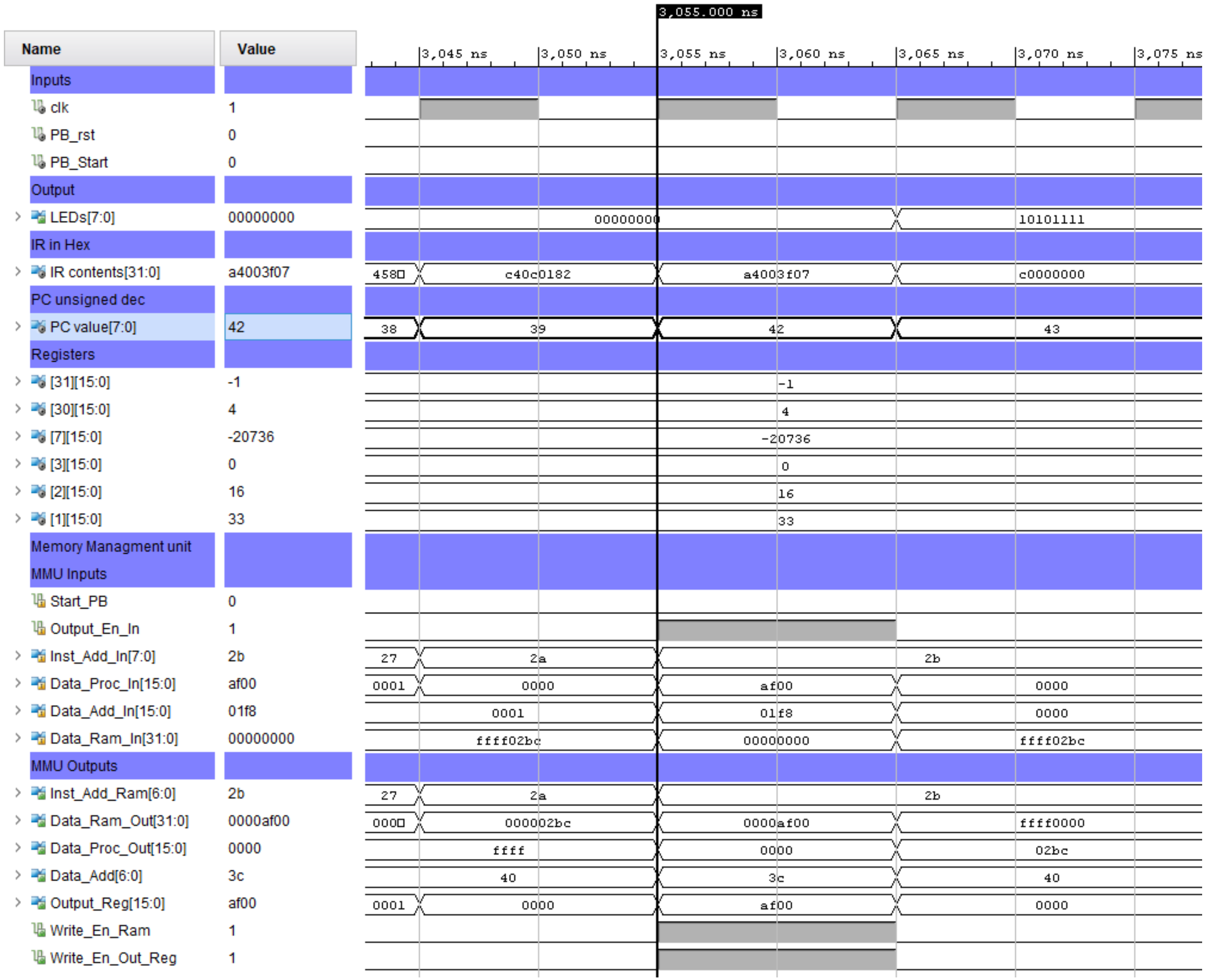
1,835.000 ns
1,820 ns
1,825 ns
1,830 ns
1,835 ns
1,840 ns
1,845 ns
1,850 ns
1,855 ns
1,860 ns
1,865 ns
1,870 ns
1,875 ns
1,880 ns
00000000
0fff0000 c000fe00 c41f0283 84400200 57c00004 887e0000 909e00c0
9 10 6 11 12 13 14
0 -1
0 4
0
0
0 16
33
0a 06 0b 0c 0d 0e 0f
0000
ffff 0000 ffff 0010 0004 0007
00000000 001f0020 00000000 000f0010 001b001c 0019001a
0a 06 0b 0c 0d 0e 0f
00000000 001f0000 00000000 000f0000 001b0000 0000001a
0000 0020 0000 0010 001c 0019
3f 40 3f 48 42 43
0000

values for instruction sequences 6-11-12



Name	Value
Inputs	
clk	1
PB_rst	0
PB_Start	0
Output	
LEDs[7:0]	00000000
IR in Hex	
IR contents[31:0]	c41f0283
PC unsigned dec	
PC value[7:0]	6
Registers	
[31][15:0]	-1
[30][15:0]	0
[7][15:0]	0
[3][15:0]	0
[2][15:0]	0
[1][15:0]	33
Memory Managment unit	
MMU Inputs	
Start_PB	0
Output_En_In	0
Inst_Add_In[7:0]	0b
Data_Proc_In[15:0]	0000
Data_Add_In[15:0]	ffff
Data_Ram_In[31:0]	00000000
MMU Outputs	
Inst_Add_Ram[6:0]	0b
Data_Ram_Out[31:0]	00000000
Data_Proc_Out[15:0]	0000
Data_Add[6:0]	3f
Output_Reg[15:0]	0000
Write_En_Ram	0
Write_En_Out_Reg	0

values for instruction sequences 24-25-26



Name	Value
Inputs	
clk	1
PB_rst	0
PB_Start	0
Output	
LEDs[7:0]	00000000
IR in Hex	
IR contents[31:0]	a4003f07
PC unsigned dec	
PC value[7:0]	42
Registers	
[31][15:0]	-1
[30][15:0]	4
[7][15:0]	-20736
[3][15:0]	0
[2][15:0]	16
[1][15:0]	33
Memory Managment unit	
MMU Inputs	
Start_PB	0
Output_En_In	1
Inst_Add_In[7:0]	2b
Data_Proc_In[15:0]	af00
Data_Add_In[15:0]	01f8
Data_Ram_In[31:0]	00000000
MMU Outputs	
Inst_Add_Ram[6:0]	2b
Data_Ram_Out[31:0]	0000af00
Data_Proc_Out[15:0]	0000
Data_Add[6:0]	3c
Output_Reg[15:0]	af00
Write_En_Ram	1
Write_En_Out_Reg	1

values for instruction sequences 42-4

The “RTL Component Statistics” part of the synthesis report :

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
         2 Input     16 Bit       Adders := 3
         3 Input     16 Bit       Adders := 1
         2 Input      8 Bit       Adders := 2
         2 Input      7 Bit       Adders := 1
+---XORs :
         2 Input     16 Bit         XORs := 1
+---Registers :
                     32 Bit    Registers := 1
                     16 Bit    Registers := 33
                      8 Bit    Registers := 1
+---Muxes :
         3 Input     32 Bit        Muxes := 1
         2 Input     16 Bit        Muxes := 6
         3 Input     16 Bit        Muxes := 2
         2 Input     10 Bit        Muxes := 1
         3 Input     10 Bit        Muxes := 1
         2 Input      9 Bit        Muxes := 2
         2 Input      8 Bit        Muxes := 2
         2 Input      7 Bit        Muxes := 1
         5 Input      5 Bit        Muxes := 1
         9 Input      5 Bit        Muxes := 1
         7 Input      5 Bit        Muxes := 1
         2 Input      4 Bit        Muxes := 1
         4 Input      4 Bit        Muxes := 1
        21 Input      4 Bit        Muxes := 1
        11 Input      3 Bit        Muxes := 1
         8 Input      2 Bit        Muxes := 1
         8 Input      1 Bit        Muxes := 1
         2 Input      1 Bit        Muxes := 67
         4 Input      1 Bit        Muxes := 1
         6 Input      1 Bit        Muxes := 1
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

The XDC file:

```
create_clock -period 10.000 -name clk -waveform {0.000 5.000} -add
[get_ports clk]

set_property PACKAGE_PIN Y9 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property PACKAGE_PIN T18 [get_ports PB_Start]
set_property IOSTANDARD LVCMOS18 [get_ports PB_Start]
set_property PACKAGE_PIN R16 [get_ports PB_Rst]
set_property IOSTANDARD LVCMOS18 [get_ports PB_Rst]
set_property PACKAGE_PIN U14 [get_ports {LEDs[7]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[7]}]
set_property PACKAGE_PIN U19 [get_ports {LEDs[6]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[6]}]
set_property PACKAGE_PIN W22 [get_ports {LEDs[5]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[5]}]
set_property PACKAGE_PIN V22 [get_ports {LEDs[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[4]}]
set_property PACKAGE_PIN U21 [get_ports {LEDs[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[3]}]
set_property PACKAGE_PIN U22 [get_ports {LEDs[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[2]}]
set_property PACKAGE_PIN T21 [get_ports {LEDs[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[1]}]
set_property PACKAGE_PIN T22 [get_ports {LEDs[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {LEDs[0]}]
```