CSU22022

Project 2 – MICROCODED INSTRUCTION SET PROCESSOR

JOHN WESLEY KOMMALA

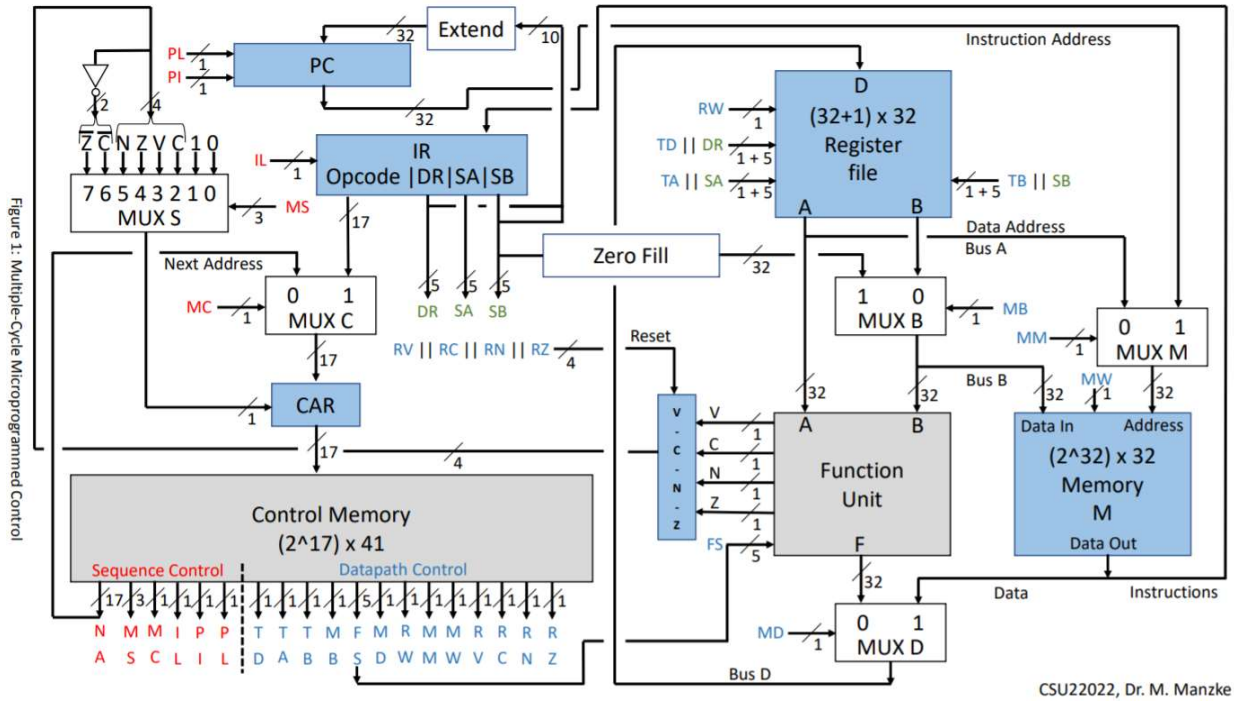
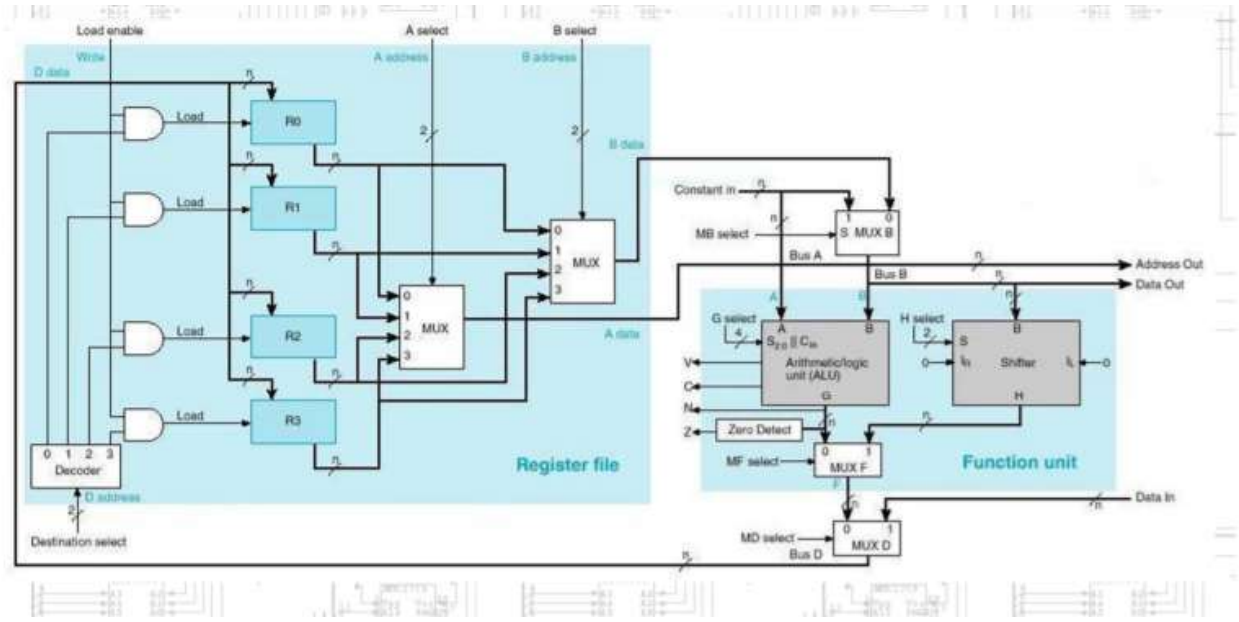


Figure 1: Multiple-Cycle Microprogrammed Control

Description: Register·file from PROJECT IA enhanced with a B-Data Bus and a Functional Unit to the 32 registers in order to obtain a 32-bit version of the Datapath shown in Figure above . The CSU22022 lecture notes has been the source of necessary information for the project.

Components



Register File 33 - 32Bit:

Components used in Register File Register 32Bit, (32 Multiplexers) MUX32_32Bit, and decoder5to32.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Register_File is
    Port (  add_A, add_B, add_D : in std_logic_vector(5 downto 0);
            Clk : in std_logic;
            load : in std_logic;
            D_out_data : in std_logic_vector(31 downto 0);
            A_out, B_out : out std_logic_vector(31 downto 0));

end Register_File;

architecture Behavioral of Register_File is
```

```
-- components
-- 32 bit Register for register file
    COMPONENT reg32
    PORT(
        D : IN std_logic_vector(31 downto 0);
        load : IN std_logic_vector(1 downto 0);
        Clk : IN std_logic;
        Q : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    -- 6 to 33 Decoder
    COMPONENT decoder_6to33
    PORT(   A0 : in std_logic;
            A1 : in std_logic;
            A2 : in std_logic;
            A3 : in std_logic;
            A4 : in std_logic;
            A5 : in std_logic;

            Q0 : out std_logic;
            Q1 : out std_logic;
            Q2 : out std_logic;
            Q3 : out std_logic;
            Q4 : out std_logic;
            Q5 : out std_logic;
            Q6 : out std_logic;
            Q7 : out std_logic;
            Q8 : out std_logic;
            Q9 : out std_logic;
            Q10 : out std_logic;
            Q11 : out std_logic;
            Q12 : out std_logic;
            Q13 : out std_logic;
            Q14 : out std_logic;
            Q15 : out std_logic;
            Q16 : out std_logic;
            Q17 : out std_logic;
            Q18 : out std_logic;
            Q19 : out std_logic;
            Q20 : out std_logic;
            Q21 : out std_logic;
```

```
            Q22 : out std_logic;
            Q23 : out std_logic;
            Q24 : out std_logic;
            Q25 : out std_logic;
            Q26 : out std_logic;
            Q27 : out std_logic;
            Q28 : out std_logic;
            Q29 : out std_logic;
            Q30 : out std_logic;
            Q31 : out std_logic;
            Q32 : out std_logic);
    END COMPONENT;

-- 33 to 1 line multiplexer (A_BUS, B_BUS)
    COMPONENT MUX_33_32Bit
    PORT(  In0, In1, In2, In3, In4, In5, In6, In7, In8, In9, In10,
           In11, In12, In13, In14, In15, In16, In17, In18, In19, In20,
           In21, In22, In23, In24, In25, In26, In27, In28, In29, In30,
           In31, In32 : in std_logic_vector(31 downto 0);

           S0, S1, S2, S3, S4, S5 : in std_logic;
           Z : out std_logic_vector(31 downto 0));
    END COMPONENT;

    -- signals
signal load_reg0,load_reg1, load_reg2, load_reg3, load_reg4, load_reg5,
load_reg6, load_reg7, load_reg8,
       load_reg9, load_reg10, load_reg11, load_reg12, load_reg13,
load_reg14, load_reg15, load_reg16,
       load_reg17, load_reg18, load_reg19, load_reg20, load_reg21,
load_reg22,load_reg23, load_reg24,
       load_reg25, load_reg26, load_reg27, load_reg28, load_reg29,
load_reg30, load_reg31, load_reg32 : std_logic:='0';

signal reg0_q, reg1_q, reg2_q, reg3_q, reg4_q, reg5_q, reg6_q, reg7_q,
reg8_q, reg9_q, reg10_q,
       reg11_q, reg12_q, reg13_q, reg14_q, reg15_q, reg16_q, reg17_q,
reg18_q, reg19_q, reg20_q,
       reg21_q, reg22_q, reg23_q, reg24_q, reg25_q, reg26_q, reg27_q,
reg28_q, reg29_q, reg30_q,
       reg31_q, reg32_q, out_sig_A, out_sig_B : std_logic_vector(31 downto
0):= (others => '0');
```

```
begin
-- port maps ;-)
-- register 0
reg00 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg0,
            load(1) => load,
            Clk => Clk,
            Q => reg0_q
);
-- register 1
reg01: reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg1,
            load(1) => load,
            Clk => Clk,
            Q => reg1_q
);
-- register 2
reg02 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg2,
            load(1) => load,
            Clk => Clk,
            Q => reg2_q
);
-- register 3
reg03 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg3,
            load(1) => load,
            Clk => Clk,
            Q => reg3_q
);
-- register 4
reg04 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg4,
            load(1) => load,
            Clk => Clk,
            Q => reg4_q
```

```
        );
        -- register 5
        reg05: reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg5,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg5_q
        );
        -- register 6
        reg06 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg6,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg6_q
        );
        -- register 7
        reg07 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg7,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg7_q
        );
        -- register 8
        reg08: reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg8,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg8_q
        );
        -- register 9
        reg09 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg9,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg9_q
        );
        -- register 10
```

```
        reg010 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg10,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg10_q
        );
        -- register 11
        reg011: reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg11,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg11_q
        );
        -- register 12
        reg012 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg12,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg12_q
        );
        -- register 13
        reg013 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg13,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg13_q
        );
        -- register 14
        reg014 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg14,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg14_q
        );
        -- register 15
        reg015: reg32 PORT MAP(
                    D => D_out_data,
```

```
                load(0) => load_reg15,
                load(1) => load,
                Clk => Clk,
                Q => reg15_q
        );
        -- register 16
        reg016 : reg32 PORT MAP(
                D => D_out_data,
                load(0) => load_reg16,
                load(1) => load,
                Clk => Clk,
                Q => reg16_q
        );
        -- register 17
        reg017 : reg32 PORT MAP(
                D => D_out_data,
                load(0) => load_reg17,
                load(1) => load,
                Clk => Clk,
                Q => reg17_q
        );
        -- register 18
        reg018: reg32 PORT MAP(
                D => D_out_data,
                load(0) => load_reg18,
                load(1) => load,
                Clk => Clk,
                Q => reg18_q
        );
        -- register 19
        reg019 : reg32 PORT MAP(
                D => D_out_data,
                load(0) => load_reg19,
                load(1) => load,
                Clk => Clk,
                Q => reg19_q
        );
        -- register 20
        reg020 : reg32 PORT MAP(
                D => D_out_data,
                load(0) => load_reg20,
                load(1) => load,
```

```
            Clk => Clk,
            Q => reg20_q
    );
    -- register 21
    reg021: reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg21,
            load(1) => load,
            Clk => Clk,
            Q => reg21_q
    );
    -- register 22
    reg022 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg22,
            load(1) => load,
            Clk => Clk,
            Q => reg22_q
    );
    -- register 23
    reg023 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg23,
            load(1) => load,
            Clk => Clk,
            Q => reg23_q
    );
    -- register 24
    reg024 : reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg24,
            load(1) => load,
            Clk => Clk,
            Q => reg24_q
    );
    -- register 25
    reg025: reg32 PORT MAP(
            D => D_out_data,
            load(0) => load_reg25,
            load(1) => load,
            Clk => Clk,
            Q => reg25_q
```

```
        );
        -- register 26
        reg026 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg26,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg26_q
        );
        -- register 27
        reg027 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg27,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg27_q
        );
        -- register 28
        reg028: reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg28,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg28_q
        );
        -- register 29
        reg029 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg29,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg29_q
        );
        -- register 30
        reg030 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg30,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg30_q
        );
        -- register 32
```

```
        reg031 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg31,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg31_q
        );
        -- register 33
        reg032 : reg32 PORT MAP(
                    D => D_out_data,
                    load(0) => load_reg32,
                    load(1) => load,
                    Clk => Clk,
                    Q => reg32_q
        );
        -- Destination register decoder
        add_Decoder_6to33: decoder_6to33 PORT MAP(
                    A0 => add_D(0),
                    A1 => add_D(1),
                    A2 => add_D(2),
                    A3 => add_D(3),
                    A4 => add_D(4),
                    A5 => add_D(5),

                    Q0 => load_reg0,
                    Q1 => load_reg1,
                    Q2 => load_reg2,
                    Q3 => load_reg3,
                    Q4 => load_reg4,
                    Q5 => load_reg5,
                    Q6 => load_reg6,
                    Q7 => load_reg7,
                    Q8 => load_reg8,
                    Q9 => load_reg9,
                    Q10 => load_reg10,
                    Q11 => load_reg11,
                    Q12 => load_reg12,
                    Q13 => load_reg13,
                    Q14 => load_reg14,
                    Q15 => load_reg15,
                    Q16 => load_reg16,
                    Q17 => load_reg17,
```

```
                Q18 => load_reg18,
                Q19 => load_reg19,
                Q20 => load_reg20,
                Q21 => load_reg21,
                Q22 => load_reg22,
                Q23 => load_reg23,
                Q24 => load_reg24,
                Q25 => load_reg25,
                Q26 => load_reg26,
                Q27 => load_reg27,
                Q28 => load_reg28,
                Q29 => load_reg29,
                Q30 => load_reg30,
                Q31 => load_reg31,
                Q32 => load_reg32
        );
        -- 33 to 1 multiplexer for A
        A_33_1_mux: MUX_33_32Bit PORT MAP(
                In0 => reg0_q,
                In1 => reg1_q,
                In2 => reg2_q,
                In3 => reg3_q,
                In4 => reg4_q,
                In5 => reg5_q,
                In6 => reg6_q,
                In7 => reg7_q,
                In8 => reg8_q,
                In9 => reg9_q,
                In10 => reg10_q,
                In11 => reg11_q,
                In12 => reg12_q,
                In13 => reg13_q,
                In14 => reg14_q,
                In15 => reg15_q,
                In16 => reg16_q,
                In17 => reg17_q,
                In18 => reg18_q,
                In19 => reg19_q,
                In20 => reg20_q,
                In21 => reg21_q,
                In22 => reg22_q,
                In23 => reg23_q,
```

```
                In24 => reg24_q,
                In25 => reg25_q,
                In26 => reg26_q,
                In27 => reg27_q,
                In28 => reg28_q,
                In29 => reg29_q,
                In30 => reg30_q,
                In31 => reg31_q,
                In32 => reg32_q,
                S0 => add_A(0),
                S1 => add_A(1),
                S2 => add_A(2),
                S3 => add_A(3),
                S4 => add_A(4),
                S5 => add_A(5),
                Z => out_sig_A

        );
        -- 33 to 1 multiplexer for B
        B_33_1_mux: MUX_33_32bit PORT MAP(
                In0 => reg0_q,
                In1 => reg1_q,
                In2 => reg2_q,
                In3 => reg3_q,
                In4 => reg4_q,
                In5 => reg5_q,
                In6 => reg6_q,
                In7 => reg7_q,
                In8 => reg8_q,
                In9 => reg9_q,
                In10 => reg10_q,
                In11 => reg11_q,
                In12 => reg12_q,
                In13 => reg13_q,
                In14 => reg14_q,
                In15 => reg15_q,
                In16 => reg16_q,
                In17 => reg17_q,
                In18 => reg18_q,
                In19 => reg19_q,
                In20 => reg20_q,
                In21 => reg21_q,
```

```
                    In22 => reg22_q,
                    In23 => reg23_q,
                    In24 => reg24_q,
                    In25 => reg25_q,
                    In26 => reg26_q,
                    In27 => reg27_q,
                    In28 => reg28_q,
                    In29 => reg29_q,
                    In30 => reg30_q,
                    In31 => reg31_q,
                    In32 => reg32_q,
                    S0 => add_B(0),
                    S1 => add_B(1),
                    S2 => add_B(2),
                    S3 => add_B(3),
                    S4 => add_B(4),
                    S5 => add_B(5),
                    Z => out_sig_B
        );

        A_out <= out_sig_A;
        B_out <= out_sig_B;

end Behavioral;
```

Register 32 Bit

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity reg32 is
    port ( D : in std_logic_vector(31 downto 0);
            load: in std_logic_vector(1 downto 0);
            Clk : in std_logic;
            Q : out std_logic_vector(31 downto 0));
end reg32;

architecture Behavioral of reg32 is
begin
process(Clk)
begin
    if (rising_edge(Clk)) then
```

```
        if load = "11" then
            Q<=D after 1 ns;
        end if;
    end if;
end process;
end Behavioral;
```

MUX33_32Bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX_33_32Bit is
    Port ( In0, In1, In2, In3, In4, In5, In6, In7, In8, In9, In10, In11, In12,
In13, In14, In15, In16, In17, In18, In19, In20, In21, In22, In23, In24, In25,
In26, In27, In28, In29, In30, In31, In32 : in std_logic_vector(31 downto 0);
            S0, S1, S2, S3, S4, S5 : in std_logic;
            Z : out std_logic_vector(31 downto 0));
end MUX_33_32Bit;

architecture Behavioral of MUX_33_32Bit is
begin
Z <= In0 after 1 ns when S0='0' and S1='0' and S2='0' and S3='0' and S4='0' and
S5='0' else
    In1 after 1 ns when S0='0' and S1='0' and S2='0' and S3='0' and S4='1' and
S5='0' else
    In2 after 1 ns when S0='0' and S1='0' and S2='0' and S3='1' and S4='0' and
S5='0' else
    In3 after 1 ns when S0='0' and S1='0' and S2='0' and S3='1' and S4='1' and
S5='0' else
    In4 after 1 ns when S0='0' and S1='0' and S2='1' and S3='0' and S4='0' and
S5='0' else
    In5 after 1 ns when S0='0' and S1='0' and S2='1' and S3='0' and S4='1' and
S5='0' else
    In6 after 1 ns when S0='0' and S1='0' and S2='1' and S3='1' and S4='0' and
S5='0' else
    In7 after 1 ns when S0='0' and S1='0' and S2='1' and S3='1' and S4='1' and
S5='0' else
    In8 after 1 ns when S0='0' and S1='1' and S2='0' and S3='0' and S4='0' and
S5='0' else
```

```
    In9 after 1 ns when S0='0' and S1='1' and S2='0' and S3='0' and S4='1' and
S5='0' else
    In10 after 1 ns when S0='0' and S1='1' and S2='0' and S3='1' and S4='0' and
S5='0' else
    In11 after 1 ns when S0='0' and S1='1' and S2='0' and S3='1' and S4='1' and
S5='0' else
    In12 after 1 ns when S0='0' and S1='1' and S2='1' and S3='0' and S4='0' and
S5='0' else
    In13 after 1 ns when S0='0' and S1='1' and S2='1' and S3='0' and S4='1' and
S5='0' else
    In14 after 1 ns when S0='0' and S1='1' and S2='1' and S3='1' and S4='0' and
S5='0' else
    In15 after 1 ns when S0='0' and S1='1' and S2='1' and S3='1' and S4='1' and
S5='0' else
    In16 after 1 ns when S0='1' and S1='0' and S2='0' and S3='0' and S4='0' and
S5='0' else
    In17 after 1 ns when S0='1' and S1='0' and S2='0' and S3='0' and S4='1' and
S5='0' else
    In18 after 1 ns when S0='1' and S1='0' and S2='0' and S3='1' and S4='0' and
S5='0' else
    In19 after 1 ns when S0='1' and S1='0' and S2='0' and S3='1' and S4='1' and
S5='0' else
    In20 after 1 ns when S0='1' and S1='0' and S2='1' and S3='0' and S4='0' and
S5='0' else
    In21 after 1 ns when S0='1' and S1='0' and S2='1' and S3='0' and S4='1' and
S5='0' else
    In22 after 1 ns when S0='1' and S1='0' and S2='1' and S3='1' and S4='0' and
S5='0' else
    In23 after 1 ns when S0='1' and S1='0' and S2='1' and S3='1' and S4='1' and
S5='0' else
    In24 after 1 ns when S0='1' and S1='1' and S2='0' and S3='0' and S4='0' and
S5='0' else
    In25 after 1 ns when S0='1' and S1='1' and S2='0' and S3='0' and S4='1' and
S5='0' else
    In26 after 1 ns when S0='1' and S1='1' and S2='0' and S3='1' and S4='0' and
S5='0' else
    In27 after 1 ns when S0='1' and S1='1' and S2='0' and S3='1' and S4='1' and
S5='0' else
    In28 after 1 ns when S0='1' and S1='1' and S2='1' and S3='0' and S4='0' and
S5='0' else
    In29 after 1 ns when S0='1' and S1='1' and S2='1' and S3='0' and S4='1' and
S5='0' else
```

```
    In30 after 1 ns when S0='1' and S1='1' and S2='1' and S3='1' and S4='0' and
S5='0' else
    In31 after 1 ns when S0='1' and S1='1' and S2='1' and S3='1' and S4='1' and
S5='0' else
    In32 after 1 ns when S0='0' and S1='0' and S2='0' and S3='0' and S4='0' and
S5='1' else
    x"00000000" after 1 ns;
end Behavioral;
```

Decoder_6to33:

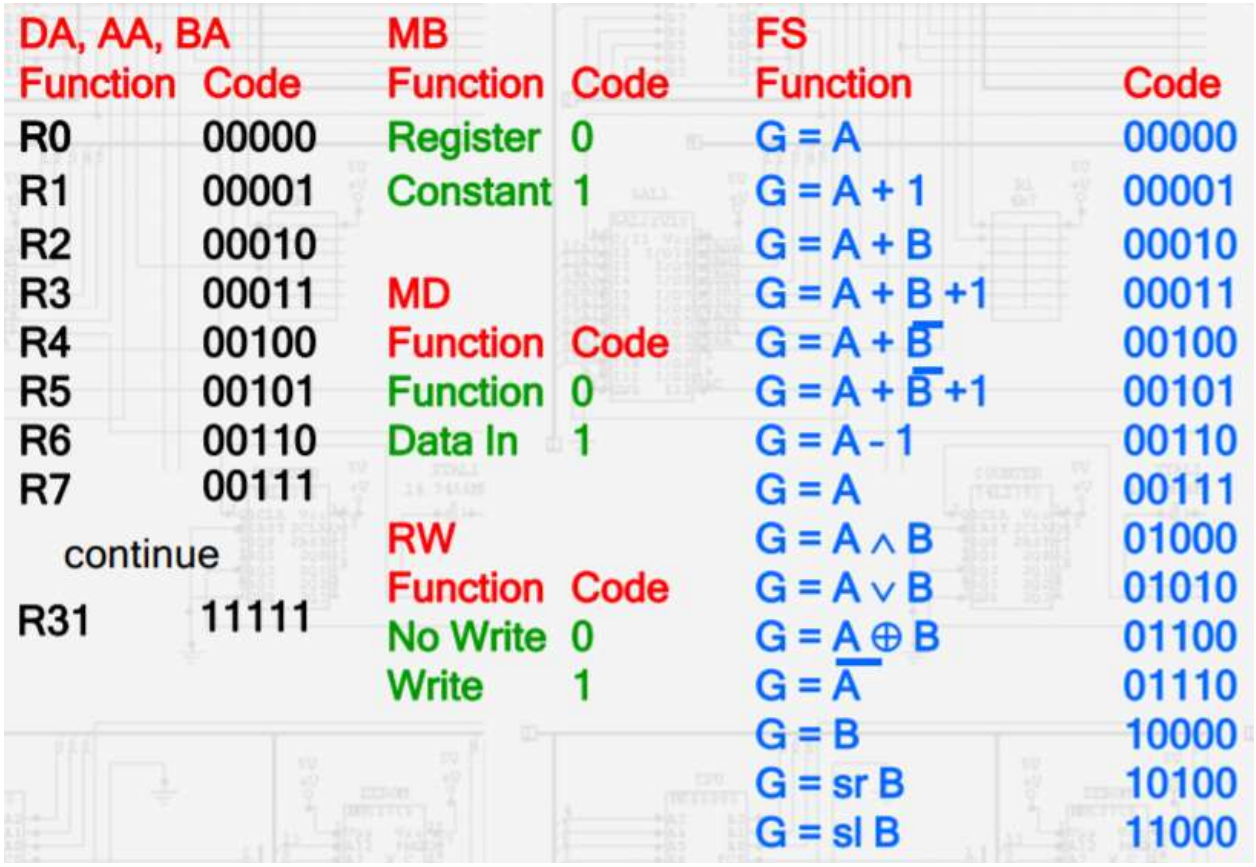
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decoder_6to33 is
    Port (  A0 : in std_logic;
            A1 : in std_logic;
            A2 : in std_logic;
            A3 : in std_logic;
            A4 : in std_logic;
            A5 : in std_logic;

            Q0 : out std_logic;
            Q1 : out std_logic;
            Q2 : out std_logic;
            Q3 : out std_logic;
            Q4 : out std_logic;
            Q5 : out std_logic;
            Q6 : out std_logic;
            Q7 : out std_logic;
            Q8 : out std_logic;
            Q9 : out std_logic;
            Q10 : out std_logic;
            Q11 : out std_logic;
            Q12 : out std_logic;
            Q13 : out std_logic;
            Q14 : out std_logic;
            Q15 : out std_logic;
            Q16 : out std_logic;
            Q17 : out std_logic;
```

```
            Q18 : out std_logic;
            Q19 : out std_logic;
            Q20 : out std_logic;
            Q21 : out std_logic;
            Q22 : out std_logic;
            Q23 : out std_logic;
            Q24 : out std_logic;
            Q25 : out std_logic;
            Q26 : out std_logic;
            Q27 : out std_logic;
            Q28 : out std_logic;
            Q29 : out std_logic;
            Q30 : out std_logic;
            Q31 : out std_logic;
            Q32 : out std_logic);
end decoder_6to33;

architecture Behavioral of decoder_6to33 is

begin

Q0 <= ((not A0) and (not A1) and (not A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --000000
Q1 <= ((not A0) and (not A1) and (not A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --000010
Q2 <= ((not A0) and (not A1) and (not A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --000100
Q3 <= ((not A0) and (not A1) and (not A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --000110
Q4 <= ((not A0) and (not A1) and (    A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --001000
Q5 <= ((not A0) and (not A1) and (    A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --001010
Q6 <= ((not A0) and (not A1) and (    A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --001100
Q7 <= ((not A0) and (not A1) and (    A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --001110
Q8 <= ((not A0) and (    A1) and (not A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --010000
Q9 <= ((not A0) and (    A1) and (not A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --010010
Q10<= ((not A0) and (    A1) and (not A2) and (    A3) and (not A4) and
```

```
(not A5)) after 1 ns; --010100
Q11<= ((not A0) and (    A1) and (not A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --010110
Q12<= ((not A0) and (    A1) and (    A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --011000
Q13<= ((not A0) and (    A1) and (    A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --011010
Q14<= ((not A0) and (    A1) and (    A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --011100
Q15<= ((not A0) and (    A1) and (    A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --011110
Q16<= ((    A0) and (not A1) and (not A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --100000
Q17<= ((    A0) and (not A1) and (not A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --100010
Q18<= ((    A0) and (not A1) and (not A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --100100
Q19<= ((    A0) and (not A1) and (not A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --100110
Q20<= ((    A0) and (not A1) and (    A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --101000
Q21<= ((    A0) and (not A1) and (    A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --101010
Q22<= ((    A0) and (not A1) and (    A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --101100
Q23<= ((    A0) and (not A1) and (    A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --101110
Q24<= ((    A0) and (    A1) and (not A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --110000
Q25<= ((    A0) and (    A1) and (not A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --110010
Q26<= ((    A0) and (    A1) and (not A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --110100
Q27<= ((    A0) and (    A1) and (not A2) and (    A3) and (    A4) and
(not A5)) after 1 ns; --110110
Q28<= ((    A0) and (    A1) and (    A2) and (not A3) and (not A4) and
(not A5)) after 1 ns; --111000
Q29<= ((    A0) and (    A1) and (    A2) and (not A3) and (    A4) and
(not A5)) after 1 ns; --111010
Q30<= ((    A0) and (    A1) and (    A2) and (    A3) and (not A4) and
(not A5)) after 1 ns; --111100
Q31<= ((    A0) and (    A1) and (    A2) and (    A3) and (    A4) and
```

```
(not A5)) after 1 ns; --111110
Q32<= ((not A0) and (not A1) and (not A2) and (not A3) and (not A4) and (
A5)) after 1 ns; --000001

end Behavioral;
```

Function Unit:



DA, AA, BA		MB		FS	
Function	Code	Function	Code	Function	Code
R0	00000	Register	0	G = A	00000
R1	00001	Constant	1	G = A + 1	00001
R2	00010			G = A + B	00010
R3	00011	MD		G = A + B +1	00011
R4	00100	Function	Code	G = A + $\overline{B}$	00100
R5	00101	Function	0	G = A + $\overline{B}$ +1	00101
R6	00110	Data In	1	G = A - 1	00110
R7	00111			G = A	00111
continue		RW		G = A ∧ B	01000
		Function	Code	G = A ∨ B	01010
R31	11111	No Write	0	G = A ⊕ B	01100
		Write	1	G = $\overline{A}$	01110
				G = B	10000
				G = sr B	10100
				G = sl B	11000

Components used in Function Unit are ALU, Shifter, and MUX 2 32Bit.

Operations :

- Transfer A (F Sel = 00000),
- Increment A (F Sel = 00001),
- Add A + B (F Sel = 00010),
- Add with carry (F Sel 00011),
- Add with not B (F Sel 00100),
- Subtract A - B (F Sel 00101),
- Decrement A - 1 (F Sel 00111),
- Transfer A (F Sel = 00111),
- A ^ B (F Sel = 01000),
- A V B (F Sel = 01010),
- A ⊕ B (F Sel 01100),
- Not A (F Sel 01110),
- TRANSFER B (F Sel = 10000),
- SHIFT RIGHT B (also displays carry) (F Sel = 10100),
- SHIFT LEFT B (also displays carry) (F Sel = 11000)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Function_Unit is
    Port ( A : in  STD_LOGIC_VECTOR(31 downto 0);
           B : in  STD_LOGIC_VECTOR(31 downto 0);
           FSel : in  STD_LOGIC_VECTOR(4 downto 0);
           C : out  STD_LOGIC;
           V : out  STD_LOGIC;
           N : out  STD_LOGIC;
           Z : out  STD_LOGIC;
           F : out  STD_LOGIC_VECTOR(31 downto 0));
end Function_Unit;

architecture Behavioral of Function_Unit is
-- Components
    -- ALU
    COMPONENT ALU_32Bit
    PORT( A : in  STD_LOGIC_VECTOR(31 downto 0);
          B : in  STD_LOGIC_VECTOR(31 downto 0);
          GSel : in  STD_LOGIC_VECTOR(3 downto 0);
          C : out  STD_LOGIC;
          V : out  STD_LOGIC;
          G : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;

    -- Shifter
    COMPONENT Shifter_32Bit
    PORT( B : in  STD_LOGIC_VECTOR(31 downto 0);
          S : in  STD_LOGIC_VECTOR(1 downto 0);
          IR : in  STD_LOGIC;
          IL : in  STD_LOGIC;
          H : out  STD_LOGIC_VECTOR(31 downto 0));
  END COMPONENT;

  -- MUX2
  COMPONENT MUX_2_32Bit
  PORT( IN0 : in  STD_LOGIC_VECTOR(31 downto 0);
        IN1 : in  STD_LOGIC_VECTOR(31 downto 0);
        S : in  STD_LOGIC;
```

```
        Z : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;

    signal ALUoutput, ShifterOutput, muxFOUT: STD_LOGIC_VECTOR(31 downto
0);

begin
-- Port Maps
    -- ALU
    ALU: ALU_32Bit
    PORT MAP(   A => A,
                B => B,
                GSel => FSel(3 downto 0),
                C => C,
                V => V,
                G => ALUoutput);

    -- Shifter
    SHIFTER: Shifter_32Bit
    PORT MAP(   B => B,
                S => FSel(3 downto 2),
                IR => '0',
                IL => '0',
                H => ShifterOutput);

    -- MUXF
    MUXF: MUX_2_32Bit
    PORT MAP(   IN0 => ALUoutput,
                IN1 => ShifterOutput,
                S => FSel(4),
                Z => muxFOUT);

    Z <= '1' when muxFOUT = x"00000000" else '0';
    N <= '1' when muxFOUT(31) = '1' else '0';

    F <= muxFOUT;
end Behavioral;
```

ALU:

The components used in ALU are Arithmetic Unit and Logic Unit.

Arithmetic Operations :

- G = A
- G = A + 1
- G = A + B
- G = A + B + 1
- G = A + notB
- G = A - B
- G = A - 1
- G = A

Logical Operations :

- G = A and B,
- G = A or B,
- G = A xor B,
- G = notA.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU_32bit is
    Port ( A : in  STD_LOGIC_VECTOR(31 downto 0);
           B : in  STD_LOGIC_VECTOR(31 downto 0);
           GSel : in  STD_LOGIC_VECTOR(3 downto 0);
           C : out  STD_LOGIC;
           V : out  STD_LOGIC;
           G : out  STD_LOGIC_VECTOR(31 downto 0));
end ALU_32bit;

architecture Behavioral of ALU_32bit is
-- Components
    -- Arithmetic Circuit
    COMPONENT Arithmetic_Unit
        PORT( A : in  STD_LOGIC_VECTOR(31 downto 0);
              B : in  STD_LOGIC_VECTOR(31 downto 0);
              Cin : in  STD_LOGIC;
              SelB : in  STD_LOGIC_VECTOR(1 downto 0);
              G : out  STD_LOGIC_VECTOR(31 downto 0);
              Cout : out  STD_LOGIC;
              V_out : out  STD_LOGIC);
    END COMPONENT;

    -- Logic Circuit
```

```
    COMPONENT Logic_Unit
        PORT(A : in  STD_LOGIC_VECTOR(31 downto 0);
             B : in  STD_LOGIC_VECTOR(31 downto 0);
             S : in  STD_LOGIC_VECTOR(1 downto 0);
             G : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;

    -- Select Arithmetic or Logic MUX
    COMPONENT MUX_2_32Bit
        PORT(S : in  STD_LOGIC;
             IN0 : in  STD_LOGIC_VECTOR (31 downto 0);
             IN1 : in  STD_LOGIC_VECTOR (31 downto 0);
             Z : out  STD_LOGIC_VECTOR (31 downto 0));
    END COMPONENT;

    signal Cin, C_out, V_out: STD_LOGIC;
    signal FSel: STD_LOGIC_VECTOR(2 downto 0);
    signal AUtoMUX, LUtoMUX: STD_LOGIC_VECTOR(31 downto 0);
begin
-- Port Maps
    -- Signals
    Cin <= GSel(0);
    FSel <= GSel(3 downto 1);

    -- Arithmetic Unit
    AU: Arithmetic_Unit PORT MAP(
        A => A,
        B => B,
        Cin => Cin,
        SelB(0) => FSel(0),
        SelB(1) => FSel(1),
        G => AUtoMUX,
        Cout => C_out,
        V_out => V_out);

    -- Logic Unit
    LU: Logic_Unit PORT MAP(
        A => A,
        B => B,
        S(0) => FSel(0),
        S(1) => FSel(1),
        G => LUtoMUX );
```

```
    -- MUX
    ALUMUX: MUX_2_32Bit PORT MAP(
        S => FSel(2),
        IN0 => AUtoMUX,
        IN1 => LUtoMUX,
        Z => G );

    -- Control Flags
    C <= C_out when FSel(2) = '0' else '0';
    V <= V_out when FSel(2) = '0' else '0';
end Behavioral;
```

Arithmetic Unit:

The components used in Arithmetic Unit are B Input Logic and Ripple Adder.

Operations:

- G = A (Transfer)
- G = A + 1 or A++ (Increment)
- G = A + B (Add)
- G = A + B + 1
- G = A + not(B)
- G = A + not(B) + 1 (Subtract)
- G = A - 1 or A-- (Decrement)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Arithmetic_Unit is
    Port ( A : in  STD_LOGIC_VECTOR(31 downto 0);
           B : in  STD_LOGIC_VECTOR(31 downto 0);
           Cin : in  STD_LOGIC;
           SelB : in  STD_LOGIC_VECTOR(1 downto 0);
           G : out  STD_LOGIC_VECTOR(31 downto 0);
           Cout : out  STD_LOGIC;
           V_out : out  STD_LOGIC);
end Arithmetic_Unit;

architecture Behavioral of Arithmetic_Unit is
-- Components
```

```
    -- B Input Logic
    COMPONENT B_Input_Logic
    PORT ( B : in  STD_LOGIC_VECTOR(31 downto 0);
           S : in  STD_LOGIC_VECTOR(1 downto 0);
           Y : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;

    -- 16 bit Ripple Adder
    COMPONENT Ripple_Adder
    PORT (A, B : in STD_LOGIC_VECTOR(31 downto 0);
        Cin : in STD_LOGIC;
        Cout, V_out : out STD_LOGIC;
        G_out : out STD_LOGIC_VECTOR(31 downto 0)
    );
    END COMPONENT;

    signal blogic_to_adder, adder_to_G_output: STD_LOGIC_VECTOR(31 downto
0):= (others => '0');
begin
-- Port Maps
    -- B Logic
    BLOGIC: B_Input_Logic PORT MAP(
                    B => B,
                    S => SelB,
                    Y => blogic_to_adder
                    );

    -- Ripple Adder
    RA: Ripple_Adder PORT MAP(
                    A => A,
                    B => blogic_to_adder,
                    Cin => Cin,
                    V_out => V_out,
                    Cout => Cout,
                    G_out => adder_to_G_output
                    );

    G <= adder_to_G_output after 2ns;
end Behaviora
```

B Input Logic:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity B_Input_Logic is
    Port ( B : in  STD_LOGIC_VECTOR(31 downto 0);
           S : in  STD_LOGIC_VECTOR(1 downto 0);
           Y : out  STD_LOGIC_VECTOR(31 downto 0));
end B_Input_Logic;

architecture Behavioral of B_Input_Logic is

begin
Y(0) <= (B(0) and S(0)) or (not B(0) and S(1)) after 1ns;
Y(1) <= (B(1) and S(0)) or (not B(1) and S(1)) after 1ns;
Y(2) <= (B(2) and S(0)) or (not B(2) and S(1)) after 1ns;
Y(3) <= (B(3) and S(0)) or (not B(3) and S(1)) after 1ns;
Y(4) <= (B(4) and S(0)) or (not B(4) and S(1)) after 1ns;
Y(5) <= (B(5) and S(0)) or (not B(5) and S(1)) after 1ns;
Y(6) <= (B(6) and S(0)) or (not B(6) and S(1)) after 1ns;
Y(7) <= (B(7) and S(0)) or (not B(7) and S(1)) after 1ns;
Y(8) <= (B(8) and S(0)) or (not B(8) and S(1)) after 1ns;
Y(9) <= (B(9) and S(0)) or (not B(9) and S(1)) after 1ns;
Y(10) <= (B(10) and S(0)) or (not B(10) and S(1)) after 1ns;
Y(11) <= (B(11) and S(0)) or (not B(11) and S(1)) after 1ns;
Y(12) <= (B(12) and S(0)) or (not B(12) and S(1)) after 1ns;
Y(13) <= (B(13) and S(0)) or (not B(13) and S(1)) after 1ns;
Y(14) <= (B(14) and S(0)) or (not B(14) and S(1)) after 1ns;
Y(15) <= (B(15) and S(0)) or (not B(15) and S(1)) after 1ns;
Y(16) <= (B(16) and S(0)) or (not B(16) and S(1)) after 1ns;
Y(17) <= (B(17) and S(0)) or (not B(17) and S(1)) after 1ns;
Y(18) <= (B(18) and S(0)) or (not B(18) and S(1)) after 1ns;
Y(19) <= (B(19) and S(0)) or (not B(19) and S(1)) after 1ns;
Y(20) <= (B(20) and S(0)) or (not B(20) and S(1)) after 1ns;
Y(21) <= (B(21) and S(0)) or (not B(21) and S(1)) after 1ns;
Y(22) <= (B(22) and S(0)) or (not B(22) and S(1)) after 1ns;
Y(23) <= (B(23) and S(0)) or (not B(23) and S(1)) after 1ns;
Y(24) <= (B(24) and S(0)) or (not B(24) and S(1)) after 1ns;
Y(25) <= (B(25) and S(0)) or (not B(25) and S(1)) after 1ns;
```

```
Y(26) <= (B(26) and S(0)) or (not B(26) and S(1)) after 1ns;
Y(27) <= (B(27) and S(0)) or (not B(27) and S(1)) after 1ns;
Y(28) <= (B(28) and S(0)) or (not B(28) and S(1)) after 1ns;
Y(29) <= (B(29) and S(0)) or (not B(29) and S(1)) after 1ns;
Y(30) <= (B(30) and S(0)) or (not B(30) and S(1)) after 1ns;
Y(31) <= (B(31) and S(0)) or (not B(31) and S(1)) after 1ns;

end Behavioral;
```

Ripple Adder:

The component used in Ripple Adder is Full Adder.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Ripple_Adder is
    Port(
        A, B : in STD_LOGIC_VECTOR(31 downto 0);
        Cin : in STD_LOGIC;
        Cout, V_out : out STD_LOGIC;
        G_out : out STD_LOGIC_VECTOR(31 downto 0)
    );
end Ripple_Adder;

architecture Behavioral of Ripple_Adder is
    Component Full_Adder
        Port(
            X, Y, Cin : in STD_LOGIC;
            Cout, Sum : out STD_LOGIC
            );
    End Component;

--signals - 32 carry bits and 1 output
signal C0, C1, C2, C3, C4, C5, C6, C7, C8, C9,
        C10, C11, C12, C13, C14, C15, C16, C17, C18, C19,
        C20, C21, C22, C23, C24, C25, C26, C27, C28, C29,
        C30, C31, C_out : STD_LOGIC;
```

```
begin
    Full_Adder_00: Full_Adder PORT MAP(
        X => A(0),
        Y => B(0),
        Cin => Cin,
        Cout => C0,
        Sum => G_out(0)
        );

    Full_Adder_01: Full_Adder PORT MAP(
        X => A(1),
        Y => B(1),
        Cin => C0,
        Cout => C1,
        Sum => G_out(1)
        );

    Full_Adder_02: Full_Adder PORT MAP(
        X => A(2),
        Y => B(2),
        Cin => C1,
        Cout => C2,
        Sum => G_out(2)
        );

    Full_Adder_03: Full_Adder PORT MAP(
        X => A(3),
        Y => B(3),
        Cin => C2,
        Cout => C3,
        Sum => G_out(3)
        );

    Full_Adder_04: Full_Adder PORT MAP(
        X => A(4),
        Y => B(4),
        Cin => C3,
        Cout => C4,
        Sum => G_out(4)
        );

    Full_Adder_05: Full_Adder PORT MAP(
```

```
        X => A(5),
        Y => B(5),
        Cin => C4,
        Cout => C5,
        Sum => G_out(5)
        );

    Full_Adder_06: Full_Adder PORT MAP(
        X => A(6),
        Y => B(6),
        Cin => C5,
        Cout => C6,
        Sum => G_out(6)
        );

    Full_Adder_07: Full_Adder PORT MAP(
        X => A(7),
        Y => B(7),
        Cin => C6,
        Cout => C7,
        Sum => G_out(7)
        );

    Full_Adder_08: Full_Adder PORT MAP(
        X => A(8),
        Y => B(8),
        Cin => C7,
        Cout => C8,
        Sum => G_out(8)
        );

    Full_Adder_09: Full_Adder PORT MAP(
        X => A(9),
        Y => B(9),
        Cin => C8,
        Cout => C9,
        Sum => G_out(9)
        );

    Full_Adder_10: Full_Adder PORT MAP(
        X => A(10),
        Y => B(10),
```

```
        Cin => C9,
        Cout => C10,
        Sum => G_out(10)
        );

    Full_Adder_11: Full_Adder PORT MAP(
        X => A(11),
        Y => B(11),
        Cin => C10,
        Cout => C11,
        Sum => G_out(11)
        );

    Full_Adder_12: Full_Adder PORT MAP(
        X => A(12),
        Y => B(12),
        Cin => C11,
        Cout => C12,
        Sum => G_out(12)
        );

    Full_Adder_13: Full_Adder PORT MAP(
        X => A(13),
        Y => B(13),
        Cin => C12,
        Cout => C13,
        Sum => G_out(13)
        );

    Full_Adder_14: Full_Adder PORT MAP(
        X => A(14),
        Y => B(14),
        Cin => C13,
        Cout => C14,
        Sum => G_out(14)
        );

    Full_Adder_15: Full_Adder PORT MAP(
        X => A(15),
        Y => B(15),
        Cin => C14,
        Cout => C15,
```

```
        Sum => G_out(15)
        );

     Full_Adder_16: Full_Adder PORT MAP(
        X => A(16),
        Y => B(16),
        Cin => C15,
        Cout => C16,
        Sum => G_out(16)
        );

    Full_Adder_17: Full_Adder PORT MAP(
        X => A(17),
        Y => B(17),
        Cin => C16,
        Cout => C17,
        Sum => G_out(17)
        );

    Full_Adder_18: Full_Adder PORT MAP(
        X => A(18),
        Y => B(18),
        Cin => C17,
        Cout => C18,
        Sum => G_out(18)
        );

    Full_Adder_19: Full_Adder PORT MAP(
        X => A(19),
        Y => B(19),
        Cin => C18,
        Cout => C19,
        Sum => G_out(19)
        );

    Full_Adder_20: Full_Adder PORT MAP(
        X => A(10),
        Y => B(10),
        Cin => C19,
        Cout => C20,
        Sum => G_out(20)
        );
```

```
    Full_Adder_21: Full_Adder PORT MAP(
        X => A(21),
        Y => B(21),
        Cin => C20,
        Cout => C21,
        Sum => G_out(21)
        );

    Full_Adder_22: Full_Adder PORT MAP(
        X => A(22),
        Y => B(22),
        Cin => C21,
        Cout => C22,
        Sum => G_out(22)
        );

    Full_Adder_23: Full_Adder PORT MAP(
        X => A(23),
        Y => B(23),
        Cin => C22,
        Cout => C23,
        Sum => G_out(23)
        );

    Full_Adder_24: Full_Adder PORT MAP(
        X => A(24),
        Y => B(24),
        Cin => C23,
        Cout => C24,
        Sum => G_out(24)
        );

    Full_Adder_25: Full_Adder PORT MAP(
        X => A(25),
        Y => B(25),
        Cin => C24,
        Cout => C25,
        Sum => G_out(25)
        );

    Full_Adder_26: Full_Adder PORT MAP(
```

```
        X => A(26),
        Y => B(26),
        Cin => C25,
        Cout => C26,
        Sum => G_out(26)
        );

    Full_Adder_27: Full_Adder PORT MAP(
        X => A(27),
        Y => B(27),
        Cin => C26,
        Cout => C27,
        Sum => G_out(27)
        );

    Full_Adder_28: Full_Adder PORT MAP(
        X => A(28),
        Y => B(28),
        Cin => C27,
        Cout => C28,
        Sum => G_out(28)
        );

    Full_Adder_29: Full_Adder PORT MAP(
        X => A(29),
        Y => B(29),
        Cin => C28,
        Cout => C29,
        Sum => G_out(29)
        );

    Full_Adder_30: Full_Adder PORT MAP(
        X => A(30),
        Y => B(30),
        Cin => C29,
        Cout => C30,
        Sum => G_out(30)
        );

    Full_Adder_31: Full_Adder PORT MAP(
        X => A(31),
        Y => B(31),
```

```
        Cin => C30,
        Cout => C31,
        Sum => G_out(31)
        );


      Cout <= C31;
        V_out <= (C30 xor C31);

end Behavioral;
```

Full Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Full_Adder is
    Port ( X : in  STD_LOGIC;
           Y : in  STD_LOGIC;
           Cin : in  STD_LOGIC;
           Sum : out  STD_LOGIC;
           Cout : out  STD_LOGIC);
end Full_Adder;

architecture Behavioral of Full_Adder is

begin
    Sum <= X xor Y xor Cin after 1ns;
    Cout <= (X and Y) or (Cin and (X xor Y)) after 1ns;
end Behavioral;
```

Logic Unit:

The component used in Logic Unit is MUX 4 32Bit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Logic_Unit is
    Port ( A : in  STD_LOGIC_VECTOR(31 downto 0);
           B : in  STD_LOGIC_VECTOR(31 downto 0);
           S : in  STD_LOGIC_VECTOR(1 downto 0);
           G : out  STD_LOGIC_VECTOR(31 downto 0));
end Logic_Unit;

architecture Behavioral of Logic_Unit is
-- Components
    -- MUX 4 to 1
    COMPONENT MUX_4_32Bit
        PORT(   IN0 : in  STD_LOGIC_VECTOR(31 downto 0);
                IN1 : in  STD_LOGIC_VECTOR(31 downto 0);
                IN2 : in  STD_LOGIC_VECTOR(31 downto 0);
                IN3 : in  STD_LOGIC_VECTOR(31 downto 0);
                S : in  STD_LOGIC_VECTOR(1 downto 0);
                Z : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;
    signal AandB, AorB, AxorB, notA: STD_LOGIC_VECTOR(31 downto 0);
begin
-- Port maps
    -- signals
    AandB <= A and B;
    AorB <= A or B;
    AxorB <= A xor B;
    notA <= not A;

    -- MUX
    MUX: MUX_4_32Bit PORT MAP(
        IN0 => AandB,
        IN1 => AorB,
        IN2 => AxorB,
        IN3 => notA,
        S => S,
        Z => G
        );

end Behavioral;
```

MUX4 32Bit:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX_4_32Bit is
    Port ( In0 : in  STD_LOGIC_VECTOR(31 downto 0);
           In1 : in  STD_LOGIC_VECTOR(31 downto 0);
           In2 : in  STD_LOGIC_VECTOR(31 downto 0);
           In3 : in  STD_LOGIC_VECTOR(31 downto 0);
           S : in  STD_LOGIC_VECTOR(1 downto 0);
           Z : out  STD_LOGIC_VECTOR(31 downto 0));
end MUX_4_32Bit;

architecture Behavioral of MUX_4_32Bit is

begin
Z <=  In0 after 1ns when S(0)='0' and S(1)='0' else
      In1 after 1ns when S(0)='1' and S(1)='0' else
      In2 after 1ns when S(0)='0' and S(1)='1' else
      In3 after 1ns when S(0)='1' and S(1)='1' else
      x"00000000" after 1ns;

end Behavioral;
```

Shifter:

The component used in Shifter is MUX 3 1Bit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Shifter_32Bit is
    Port(
        B : in STD_LOGIC_VECTOR(31 downto 0);
        S : in STD_LOGIC_VECTOR(1 downto 0);
        IL, IR : in STD_LOGIC;
        H : out STD_LOGIC_VECTOR(31 downto 0)
        );
end Shifter_32Bit;
```

```
architecture Behavioral of Shifter_32Bit is
--3 to 1 mux
    Component MUX_3_1Bit
        Port(
            In0, In1, In2, S0, S1 : in STD_LOGIC;
            Z : out STD_LOGIC
        );
    End Component;
begin
    mux00: MUX_3_1Bit PORT MAP(
        In0 => B(0),
        In1 => B(1),
        In2 => IL,
        S0 => S(0),
        S1 => S(1),
        Z => H(0)
    );

    mux01: MUX_3_1Bit PORT MAP(
        In0 => B(1),
        In1 => B(2),
        In2 => B(0),
        S0 => S(0),
        S1 => S(1),
        Z => H(1)
    );

    mux02: MUX_3_1Bit PORT MAP(
        In0 => B(2),
        In1 => B(3),
        In2 => B(1),
        S0 => S(0),
        S1 => S(1),
        Z => H(2)
    );

    mux03: MUX_3_1Bit PORT MAP(
        In0 => B(3),
        In1 => B(4),
        In2 => B(2),
        S0 => S(0),
        S1 => S(1),
```

```
        Z => H(3)
    );

    mux04: MUX_3_1Bit PORT MAP
    (
        In0 => B(4),
        In1 => B(5),
        In2 => B(3),
        S0 => S(0),
        S1 => S(1),
        Z => H(4)
    );

    mux05: MUX_3_1Bit PORT MAP
    (
        In0 => B(5),
        In1 => B(6),
        In2 => B(4),
        S0 => S(0),
        S1 => S(1),
        Z => H(5)
    );

    mux06: MUX_3_1Bit PORT MAP
    (
        In0 => B(6),
        In1 => B(7),
        In2 => B(5),
        S0 => S(0),
        S1 => S(1),
        Z => H(6)
    );

    mux07: MUX_3_1Bit PORT MAP
    (
        In0 => B(7),
        In1 => B(8),
        In2 => B(6),
        S0 => S(0),
        S1 => S(1),
        Z => H(7)
    );
```

```
    mux08: MUX_3_1Bit PORT MAP
    (
        In0 => B(8),
        In1 => B(9),
        In2 => B(7),
        S0 => S(0),
        S1 => S(1),
        Z => H(8)
    );

    mux09: MUX_3_1Bit PORT MAP
    (
        In0 => B(9),
        In1 => B(10),
        In2 => B(8),
        S0 => S(0),
        S1 => S(1),
        Z => H(9)
    );

    mux10: MUX_3_1Bit PORT MAP
    (
        In0 => B(10),
        In1 => B(11),
        In2 => B(9),
        S0 => S(0),
        S1 => S(1),
        Z => H(10)
    );

    mux11: MUX_3_1Bit PORT MAP
    (
        In0 => B(11),
        In1 => B(12),
        In2 => B(10),
        S0 => S(0),
        S1 => S(1),
        Z => H(11)
    );

    mux12: MUX_3_1Bit PORT MAP
```

```
    (
        In0 => B(12),
        In1 => B(13),
        In2 => B(11),
        S0 => S(0),
        S1 => S(1),
        Z => H(12)
    );

    mux13: MUX_3_1Bit PORT MAP
    (
        In0 => B(13),
        In1 => B(14),
        In2 => B(12),
        S0 => S(0),
        S1 => S(1),
        Z => H(13)
    );

    mux14: MUX_3_1Bit PORT MAP
    (
        In0 => B(14),
        In1 => B(15),
        In2 => B(13),
        S0 => S(0),
        S1 => S(1),
        Z => H(14)
    );

    mux15: MUX_3_1Bit PORT MAP
    (
        In0 => B(15),
        In1 => B(16),
        In2 => B(14),
        S0 => S(0),
        S1 => S(1),
        Z => H(15)
    );

    mux16: MUX_3_1Bit PORT MAP
    (
        In0 => B(16),
```

```
        In1 => B(17),
        In2 => B(15),
        S0 => S(0),
        S1 => S(1),
        Z => H(16)
    );

    mux17: MUX_3_1Bit PORT MAP
    (
        In0 => B(17),
        In1 => B(18),
        In2 => B(16),
        S0 => S(0),
        S1 => S(1),
        Z => H(17)
    );

    mux18: MUX_3_1Bit PORT MAP
    (
        In0 => B(18),
        In1 => B(19),
        In2 => B(17),
        S0 => S(0),
        S1 => S(1),
        Z => H(18)
    );

    mux19: MUX_3_1Bit PORT MAP
    (
        In0 => B(19),
        In1 => B(20),
        In2 => B(18),
        S0 => S(0),
        S1 => S(1),
        Z => H(19)
    );

    mux20: MUX_3_1Bit PORT MAP
    (
        In0 => B(20),
        In1 => B(21),
        In2 => B(19),
```

```
        S0 => S(0),
        S1 => S(1),
        Z => H(20)
    );

    mux21: MUX_3_1Bit PORT MAP
    (
        In0 => B(21),
        In1 => B(22),
        In2 => B(20),
        S0 => S(0),
        S1 => S(1),
        Z => H(21)
    );

    mux22: MUX_3_1Bit PORT MAP
    (
        In0 => B(22),
        In1 => B(23),
        In2 => B(21),
        S0 => S(0),
        S1 => S(1),
        Z => H(22)
    );

    mux23: MUX_3_1Bit PORT MAP
    (
        In0 => B(23),
        In1 => B(24),
        In2 => B(22),
        S0 => S(0),
        S1 => S(1),
        Z => H(23)
    );

    mux24: MUX_3_1Bit PORT MAP
    (
        In0 => B(24),
        In1 => B(25),
        In2 => B(23),
        S0 => S(0),
        S1 => S(1),
```

```
        Z => H(24)
    );

    mux25: MUX_3_1Bit PORT MAP
    (
        In0 => B(25),
        In1 => B(26),
        In2 => B(24),
        S0 => S(0),
        S1 => S(1),
        Z => H(25)
    );

    mux26: MUX_3_1Bit PORT MAP
    (
        In0 => B(26),
        In1 => B(27),
        In2 => B(25),
        S0 => S(0),
        S1 => S(1),
        Z => H(26)
    );

    mux27: MUX_3_1Bit PORT MAP
    (
        In0 => B(27),
        In1 => B(28),
        In2 => B(26),
        S0 => S(0),
        S1 => S(1),
        Z => H(27)
    );

    mux28: MUX_3_1Bit PORT MAP
    (
        In0 => B(28),
        In1 => B(29),
        In2 => B(27),
        S0 => S(0),
        S1 => S(1),
        Z => H(28)
    );
```

```
    mux29: MUX_3_1Bit PORT MAP
    (
        In0 => B(29),
        In1 => B(20),
        In2 => B(28),
        S0 => S(0),
        S1 => S(1),
        Z => H(29)
    );

    mux30: MUX_3_1Bit PORT MAP
    (
        In0 => B(30),
        In1 => B(31),
        In2 => B(29),
        S0 => S(0),
        S1 => S(1),
        Z => H(30)
    );

    mux31: MUX_3_1Bit PORT MAP
    (
        In0 => B(31),
        In1 => IR,
        In2 => B(30),
        S0 => S(0),
        S1 => S(1),
        Z => H(31)
    );

end Behavioral;
```

MUX 3 1 Bit:

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

entity MUX_3_1Bit is
    Port(
    In0, In1, In2 : in STD_LOGIC;
```

```
    S0, S1 : in STD_LOGIC;
    Z : out STD_LOGIC
    );
end MUX_3_1Bit;

architecture Behavioral of MUX_3_1Bit is
begin
    Z <= In0 after 1ns when S0 = '0' and S1 = '0' else
         In1 after 1ns when S0 = '0' and S1 = '1' else
         In2 after 1ns when S0 = '1' and S1 = '0' else
         '0' after 1ns;
end Behavioral;
```

MUX 2 32Bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX_2_32Bit is
    Port(
       In0 : IN  std_logic_vector(31 downto 0);
       In1 : IN  std_logic_vector(31 downto 0);
       s : in std_logic;
       Z : out std_logic_vector(31 downto 0)
    );
end MUX_2_32Bit;

architecture Behavioral of MUX_2_32Bit is
begin
    Z <= In0 after 5 ns when s='0' else
    In1 after 5 ns when s='1' else
    x"00000000" after 5 ns;
end Behavioral;
```

Zero Fill:


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

entity ZeroFill is
    Port( SB_in : in STD_LOGIC_VECTOR(4 downto 0);
        ZeroFill_out : out STD_LOGIC_VECTOR(31 downto 0)
        );
end ZeroFill;

architecture Behavioral of ZeroFill is
--    signal ZeroFill : STD_LOGIC_VECTOR(31 downto 0);
begin
    ZeroFill_out(4 downto 0) <= SB_in;
    ZeroFill_out(31 downto 5) <= x"000000" & "000";;
end Behavioral;
```

Datapath:

The Components used in Datapath is Register File, Function Unit and MUX 2 32Bit.

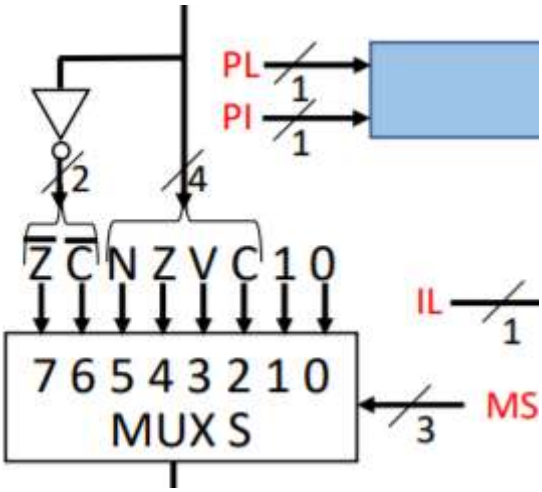
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Datapath_32Bit is
    Port (
           CONTROL_WORD : in  STD_LOGIC_VECTOR(23 downto 0);
           CONST_IN : in  STD_LOGIC_VECTOR(31 downto 0);
           DATA_IN : in  STD_LOGIC_VECTOR(31 downto 0);
           CLK : in  STD_LOGIC;
```

```
            TD, TA, TB : in STD_LOGIC;
            ADDR_OUT : out  STD_LOGIC_VECTOR(31 downto 0);
            DATA_OUT : out  STD_LOGIC_VECTOR(31 downto 0);
            status_out : out STD_LOGIC_VECTOR(3 downto 0)
            );
end Datapath_32Bit;

architecture Behavioral of Datapath_32Bit is
-- Components
    -- Regfile
    COMPONENT Register_File
    PORT(   add_A, add_B, add_D : in std_logic_vector(5 downto 0);
            Clk : in std_logic;
            load : in std_logic;
            D_out_data : in std_logic_vector(31 downto 0);
            A_out, B_out: out std_logic_vector(31 downto 0));
    END COMPONENT;

    -- Function Unit
    COMPONENT Function_Unit
    PORT( A : in  STD_LOGIC_VECTOR(31 downto 0);
          B : in  STD_LOGIC_VECTOR(31 downto 0);
          FSel : in  STD_LOGIC_VECTOR(4 downto 0);
          C : out  STD_LOGIC;
          V : out  STD_LOGIC;
          N : out  STD_LOGIC;
          Z : out  STD_LOGIC;
          F : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;

    -- Mux2
    COMPONENT MUX_2_32Bit
    PORT( IN0 : in  STD_LOGIC_VECTOR(31 downto 0);
          IN1 : in  STD_LOGIC_VECTOR(31 downto 0);
          S : in  STD_LOGIC;
          Z : out  STD_LOGIC_VECTOR(31 downto 0));
    END COMPONENT;

    COMPONENT ZeroFill is
    Port( SB_in : in STD_LOGIC_VECTOR(4 downto 0);
        ZeroFill_out : out STD_LOGIC_VECTOR(31 downto 0)
        );
```

```
    end COMPONENT;

    signal B_regfile_muxB, muxB_out, muxM_out, A_regfile_Function_Unit,
Function_Unit_out, muxD_out,
            zero_fill_out, pc_sig: STD_LOGIC_VECTOR(31 downto 0) := (others
=> '0');

    signal dest_d, addr_a, addr_b : STD_LOGIC_VECTOR(5 downto 0) ;
--
begin
-- Port Maps
-- MUXB
    MUXB: MUX_2_32Bit
    PORT MAP(   IN0 => B_regfile_muxB,
                IN1 => CONST_IN,
                S => CONTROL_WORD(8),
                Z => muxB_out);

--MUXD
    MUXD: MUX_2_32Bit
    PORT MAP(   IN0 => Function_Unit_out,
                IN1 => DATA_IN,
                S => CONTROL_WORD(2),
                Z => muxD_out);

   pc_sig <= CONST_IN;

--MUXM
    MUXM: MUX_2_32Bit
    PORT MAP(   IN0 => A_regfile_Function_Unit,
                IN1 => pc_sig,
                S => CONTROL_WORD(0),
                Z => muxM_out);

    dest_d <= TD & CONTROL_WORD(23 downto 19);
    addr_a <= TA & CONTROL_WORD(18 downto 14);
    addr_b <= TB & CONTROL_WORD(13 downto 9);

-- Zero Fill
    zero_fill: ZeroFill
    PORT MAP( SB_in => CONTROL_WORD(13 downto 9),
              ZeroFill_out  => zero_fill_out
```

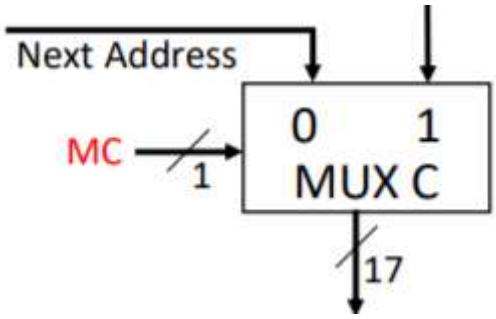
```
            );
-- Regfile
    REGFILE: Register_File
    PORT MAP(
                add_A => addr_a,
                add_B => addr_b,
                add_D => dest_d,
                CLK => CLK,
                load => CONTROL_WORD(1),
                D_out_data => muxD_out,
                A_out => A_regfile_Function_Unit,
                B_out => B_regfile_muxB
                );
   -- Function Unit
    FUNC_UNIT: Function_Unit
    PORT MAP(   A => A_regfile_Function_Unit,
                B => muxB_out,
                FSel => CONTROL_WORD(7 downto 3),
                C =>  status_out(2),
                V =>  status_out(3),
                N =>  status_out(1),
                Z =>  status_out(0),
                F => Function_Unit_out);

-- Signals
    ADDR_OUT <= muxM_out;
    DATA_OUT <= muxB_out;
end Behavioral;
```

MUX8 1Bit:



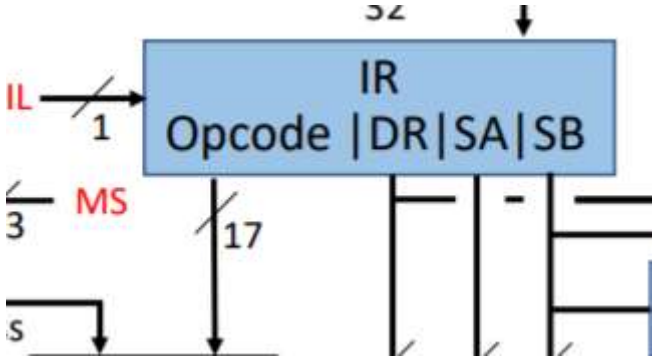

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX8_1Bit is
    Port (  In_zero, In_one, In_n, In_z, In_c, In_v, In_not_c, In_not_z :
in STD_LOGIC;
            S : in  STD_LOGIC_VECTOR(2 downto 0);
            Z : out  STD_LOGIC);
end MUX8_1Bit;

architecture Behavioral of MUX8_1Bit is

begin
    Z <= In_zero  after  1ns  when  S = "000" else
         In_one   after  1ns  when  S = "001" else
         In_c     after  1ns  when  S = "010" else
         In_v     after  1ns  when  S = "011" else
         In_z     after  1ns  when  S = "100" else
         In_n     after  1ns  when  S = "101" else
         In_not_c after  1ns  when  S = "110" else
         In_not_z after  1ns  when  S = "111";

end Behavioral;
```

MUX2 17Bit:



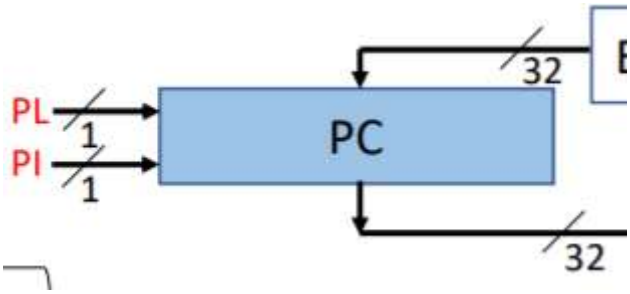

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_2_17Bit is
    Port(   In0_NA, In1_opcode : in STD_LOGIC_VECTOR(16 downto 0);
            S_mc : in STD_LOGIC;
            out_car : out STD_LOGIC_VECTOR(16 downto 0)
        );
end MUX_2_17Bit;

architecture Behavioral of MUX_2_17Bit is

begin
    out_car <= In0_NA after 1ns when S_mc='0' else
               In1_opcode after 1ns when S_mc='1' else
               "0" & x"0000" after 1ns;
end Behavioral;
```

Instruction Register:




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

entity InstructionRegister is
    Port ( INSTR : in  STD_LOGIC_VECTOR(31 downto 0);
           IL : in  STD_LOGIC;
           OPCODE : out  STD_LOGIC_VECTOR(16 downto 0);
           DR : out  STD_LOGIC_VECTOR(4 downto 0);
           SA : out  STD_LOGIC_VECTOR(4 downto 0);
           SB : out  STD_LOGIC_VECTOR(4 downto 0));
end InstructionRegister;

architecture Behavioral of InstructionRegister is
begin
    OPCODE <= INSTR(31 downto 15) after 1ns when IL = '1';
    DR <= INSTR(14 downto 10) after 1ns when IL = '1';
    SA <= INSTR(9 downto 5) after 1ns when IL = '1';
    SB <= INSTR(4 downto 0) after 1ns when IL = '1';

end Behavioral;
```

Program Counter:




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ProgrammeCounter is
    Port( PC_module_in : in STD_LOGIC_VECTOR(31 downto 0);
        PL_module_in, PI_module_in : in STD_LOGIC;
        reset, clock : in std_logic;
        PC_module_out : out STD_LOGIC_VECTOR(31 downto 0)
        );
end ProgrammeCounter;

architecture Behavioral of ProgrammeCounter is

    component ALU_32bit is
    Port ( A : in  STD_LOGIC_VECTOR(31 downto 0);
           B : in  STD_LOGIC_VECTOR(31 downto 0);
           GSel : in  STD_LOGIC_VECTOR(3 downto 0);
           C : out  STD_LOGIC;
           V : out  STD_LOGIC;
           G : out  STD_LOGIC_VECTOR(31 downto 0));
    end component;

    signal curr_address, alu_q: std_logic_vector (31 downto 0);
    signal sel: std_logic_vector (3 downto 0);

begin

    alu: ALU_32bit port map (
                            A => curr_address,
```

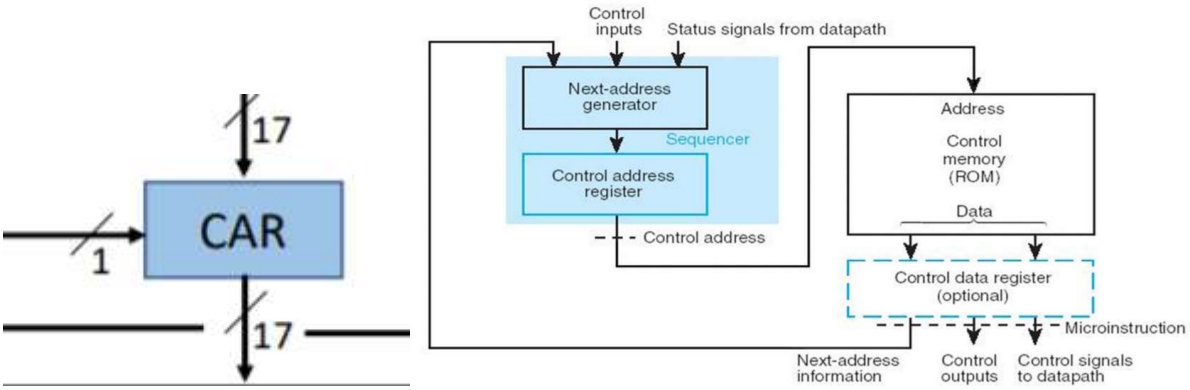
```
                            B => PC_module_in,
                            GSel => sel,
                            G => alu_q
                           );
    PC_module_out <= curr_address;

process(clock, reset)
begin
    if PL_module_in = '1' and PI_module_in = '0' then sel <= x"2" after
1ns;
    elsif PL_module_in = '0' and PI_module_in = '1' then sel <= x"1" after
1ns;
    elsif PL_module_in = '0' and PI_module_in = '0' then sel <= x"0" after
1ns;
    end if;
     if(rising_edge(clock)) then
        if reset='1' then curr_address <= x"00000000" after 1ns;
            sel <= x"0" after 1ns;
        else curr_address <= alu_q after 1ns;
        end if;
    end if;
  end process;
end Behavioral;
```

Extended Programme Counter:


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

entity ExtendedProgrammeCounter is
    Port( SR_SB : in STD_LOGIC_VECTOR(9 downto 0);
          ExtendedProgrammeCounter : out STD_LOGIC_VECTOR(31 downto 0)
        );
end ExtendedProgrammeCounter;

architecture Behavioral of ExtendedProgrammeCounter is
```

```
    signal extended_signal : STD_LOGIC_VECTOR(31 downto 0);

begin
    process(SR_SB)
    begin
        case SR_SB(9) is
            when '0' => ExtendedProgrammeCounter <= "00" & x"00000" & SR_SB
after 1ns;
            when '1' => ExtendedProgrammeCounter <= "11" & x"fffff" & SR_SB
after 1ns;
            when others => ExtendedProgrammeCounter <= x"00000000" after
1ns;
        end case;
    end process;

end Behavioral;
```

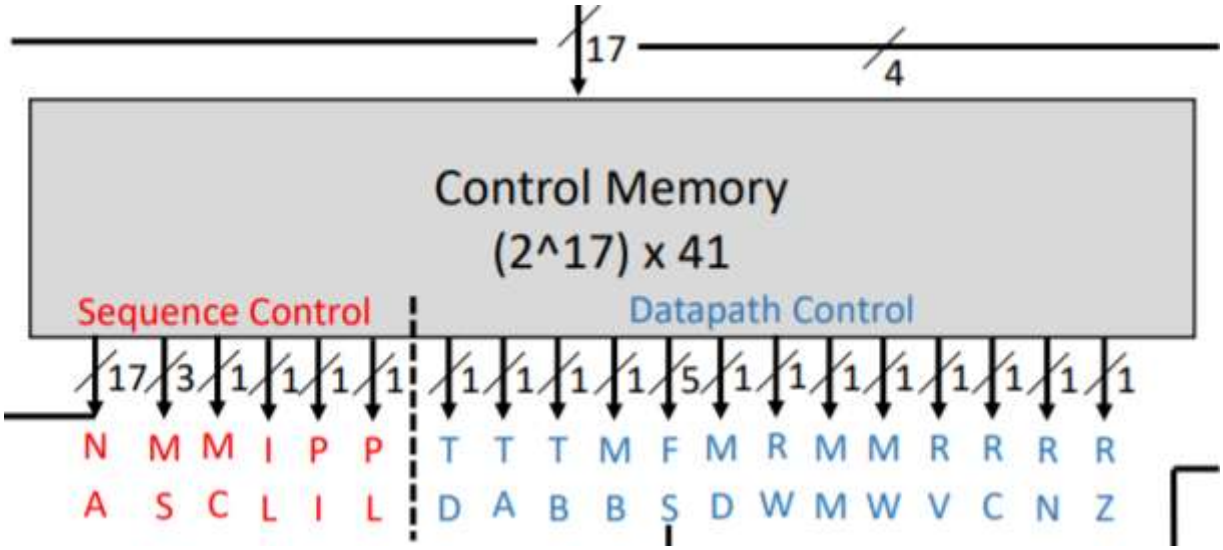
Control Address Register :




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CAR is
    Port( CAR_in : in STD_LOGIC_VECTOR(16 downto 0);
        MUX_S, reset : in STD_LOGIC;
        CAR_out : out STD_LOGIC_VECTOR(16 downto 0)
        );
end CAR;
```

```
architecture Behavioral of CAR is

    component ALU_32bit is
    Port ( A : in  STD_LOGIC_VECTOR(31 downto 0);
           B : in  STD_LOGIC_VECTOR(31 downto 0);
           GSel : in  STD_LOGIC_VECTOR(3 downto 0);
           C : out  STD_LOGIC;
           V : out  STD_LOGIC;
           G : out  STD_LOGIC_VECTOR(31 downto 0));
    end component;

    signal alu_out, alu_in: std_logic_vector (31 downto 0);
    signal curr_addr: STD_LOGIC_VECTOR(16 downto 0);
    signal sel: std_logic_vector (3 downto 0);

begin
    alu : ALU_32bit port map (
                             A => alu_in,
                             B => x"00000000",
                             GSel => sel,
                             G => alu_out
                            );

    sel <= x"1";
    CAR_out <= curr_addr;
    alu_in <= "000" & x"000" & curr_addr;

process(reset)
begin
      if reset='1' then curr_addr <= "0" & x"0000" after 1ns;
      else
        if MUX_S = '1' then curr_addr <= CAR_in after 1ns;
        else curr_addr <= alu_out(16 downto 0) after 1ns;
        end if;
      end if;
  end process;
end Behavioral;
```

Control Memory:




```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ControlMemory is
Port (  FL : out std_logic; -- 0
        RZ : out std_logic; -- 1
        RN : out std_logic; -- 2
        RC : out std_logic; -- 3
        RV : out std_logic; -- 4
        MW : out std_logic; -- 5
        MM : out std_logic; -- 6
        RW : out std_logic; -- 7
        MD : out std_logic; -- 8
        FS : out std_logic_vector(4 downto 0); -- 9 to 13
        MB : out std_logic; -- 14
        TB : out std_logic; -- 15
        TA : out std_logic; -- 16
        TD : out std_logic; -- 17
        PL : out std_logic; -- 18
        PI : out std_logic; -- 19
        IL : out std_logic; -- 20
        MC : out std_logic; -- 21
        MS : out std_logic_vector(2 downto 0); -- 22 to 24
        NA : out std_logic_vector(16 downto 0); -- 25 to 41
```

```
        IN_CAR : in std_logic_vector(16 downto 0));
end ControlMemory;

architecture Behavioral of ControlMemory is
type mem_array is array(0 to 255) of std_logic_vector(41 downto 0);

begin

memory_m: process(IN_CAR)
    variable control_mem : mem_array:=(


-- |4              2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1              5|4 2|1|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- |Next Add         | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- |Next Add         | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
  b"00000000011000000_001_0_0_0_0_0_0_0_1_00010_0_1_0_0_1_1_0_0_0", -- 00 -
ADI  D  A  B
  b"00000000011000000_001_0_0_0_0_0_0_0_0_00000_1_1_0_0_1_0_1_1_1", -- 01 -
LDR  D [A]
  b"00000000011000000_001_0_0_0_0_0_0_0_0_00000_0_0_0_1_0_1_1_1_1", -- 02 -
STR [A] B
  b"00000000011000000_001_0_0_0_0_0_0_0_0_00001_0_1_0_0_0_1_0_1_1", -- 03 -
INC  D  A
  b"00000000011000000_001_0_0_0_0_0_0_0_0_01110_0_1_0_0_0_0_1_0_1", -- 04 -
NOT  D  A
  b"00000000011000000_001_0_0_0_0_0_0_0_0_00010_0_1_0_0_0_1_0_1_1", -- 05 -
ADD  D  A  B
  b"00000000011000000_001_0_0_0_1_0_0_0_0_00000_0_0_0_0_0_0_1_1_1", -- 06 -
B    B
  b"00000000001100000_111_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_1_0_1", -- 07 -
BNE  B
  b"00000000011000000_001_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_1_0_1", -- 08 -
NOP
  b"00000000011000000_001_0_0_0_0_0_0_0_0_10100_0_1_0_0_0_0_0_0_1", -- 09 -
STR  D  B
  b"00000000011000000_001_0_0_0_0_0_0_0_0_00110_0_1_0_0_0_0_1_1_1", -- 0a -
DEC  D  A
  b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0_0", -- 0B
  b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0_0", -- 0C
  b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0_0", -- 0D
  b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0_0", -- 0E
```

```
  b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 0F

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 10
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 11
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 12
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 13
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 14
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 15
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 16
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 17
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 18
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 19
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 1A
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 1B
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 1C
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 1D
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 1E
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 1F

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 20
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 21
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 22
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 23
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 24
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 25
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 26
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 27
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 28
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 29
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 2A
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 2B
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 2C
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 2D
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 2E
```

```
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 2F

-- |4               2|2 2|2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 30
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 31
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 32
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 33
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 34
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 35
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 36
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 37
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 38
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 39
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 3A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 3B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 3C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 3D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 3E
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 3F

-- |4               2|2 2|2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 40
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 41
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 42
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 43
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 44
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 45
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 46
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 47
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 48
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 49
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 4A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 4B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 4C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 4D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 4E
```

```
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 4F

-- |4                2|2 2|2|2|2|1|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1                5|4 2|2|0|9|8|7|6|5|4|3|2   9|8|7|6|5|4|3|2|1|0|
-- | Next Address      | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address      | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 50
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 51
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 52
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 53
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 54
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 55
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 56
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 57
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 58
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 59
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 5A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 5B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 5C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 5D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 5E
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 5F

-- |4                2|2 2|2|2|2|1|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1                5|4 2|2|0|9|8|7|6|5|4|3|2   9|8|7|6|5|4|3|2|1|0|
-- | Next Address      | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address      | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 60
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 61
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 62
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 63
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 64
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 65
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 66
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 67
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 68
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 69
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 6A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 6B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 6C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 6D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 6E
```

```
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 6F

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 70
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 71
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 72
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 73
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 74
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 75
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 76
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 77
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 78
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 79
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 7A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 7B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 7C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 7D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 7E
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 7F

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 80
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 81
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 82
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 83
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 84
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 85
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 86
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 87
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 88
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 89
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 8A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 8B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 8C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 8D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 8E
```

```
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 8F

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 90
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 91
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 92
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 93
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 94
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 95
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 96
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 97
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 98
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 99
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 9A
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 9B
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 9C
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 9D
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 9E
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- 9F

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A0
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A1
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A2
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A3
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A4
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A5
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A6
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A7
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A8
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- A9
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- AA
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- AB
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- AC
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- AD
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- AE
```

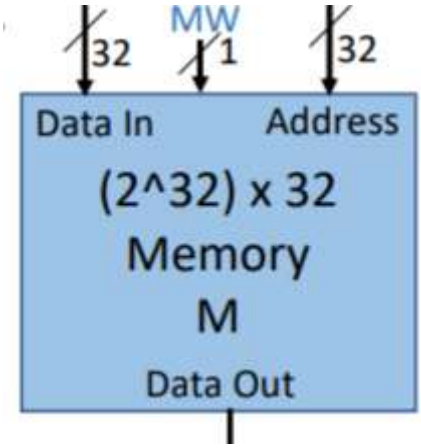
```
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- AF

-- |4                2|2 2|2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1                5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address      | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address      | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B0
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B1
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B2
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B3
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B4
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B5
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B6
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B7
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B8
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- B9
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- BA
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- BB
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- BC
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- BD
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- BE
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- BF


-- |4                2|2 2|2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1                5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address      | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address      | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"0000000011000001_000_0_1_1_0_0_0_0_0_00000_0_0_1_0_0_0_0_0", -- C0
- if fetching
    b"0000000000000000_001_1_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C1
- exit signal
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C2
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C3
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C4
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C5
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C6
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C7
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C8
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- C9
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- CA
    b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- CB
```

```
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- CC
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- CD
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- CE
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- CF

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D0
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D1
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D2
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D3
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D4
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D5
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D6
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D7
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D8
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- D9
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- DA
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- DB
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- DC
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- DD
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- DE
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- DF

-- |4               2|2 2|2|2|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1               5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address     | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address     | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E0
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E1
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E2
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E3
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E4
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E5
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E6
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E7
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E8
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- E9
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- EA
   b"0000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- EB
```

```
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- EC
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- ED
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- EE
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- EF

-- |4                2|2 2|2|2|1|1|1|1|1|1|1   0|0|0|0|0|0|0|0|0|0|
-- |1                5|4 2|2|0|9|8|7|6|5|4|3   9|8|7|6|5|4|3|2|1|0|
-- | Next Address      | MS|M|I|P|P|T|T|T|M| FS  |M|R|M|M|R|R|R|R|F|
-- | Next Address      | MS|C|L|I|L|D|A|B|B| FS  |D|W|M|W|V|C|N|Z|L|
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F0
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F1
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F2
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F3
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F4
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F5
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F6
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F7
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F8
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- F9
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- FA
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- FB
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- FC
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- FD
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0", -- FE
    b"00000000000000000_000_0_0_0_0_0_0_0_0_00000_0_0_0_0_0_0_0_0"  -- FF
     );
variable addr : integer;
variable control_out : std_logic_vector(41 downto 0);

begin
    addr := conv_integer(IN_CAR);
    control_out := control_mem(addr);
    FL <= control_out(0);
    RZ <= control_out(1);
    RN <= control_out(2);
    RC <= control_out(3);
    RV <= control_out(4);
    MW <= control_out(5);
    MM <= control_out(6);
    RW <= control_out(7);
    MD <= control_out(8);
    FS <= control_out(13 downto 9);
```

```
    MB <= control_out(14);
    TB <= control_out(15);
    TA <= control_out(16);
    TD <= control_out(12);
    PL <= control_out(17);
    PI <= control_out(19);
    IL <= control_out(20);
    MC <= control_out(21);
    MS <= control_out(24 downto 22);
    NA <= control_out(41 downto 25);
    end process;
end Behavioral;
```

Memory Module:




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memory is -- use unsigned for memory address
    Port ( address : in std_logic_vector(31 downto 0);
           write_data : in std_logic_vector(31 downto 0);
           MemWrite : in std_logic;
           clk : in std_logic;
           read_data : out std_logic_vector(31 downto 0)
           );
end memory;

architecture Behavioral of memory is
    type mem_array is array(0 to 511) of std_logic_vector(31 downto 0);
begin
    mem_process: process (address, write_data)
    variable data_mem : mem_array := (
      -- module 00
      b"0000000000000000_00001_00001_00011", -- 00 adi r1, r1, #3
      b"0000000000000000_00000_00000_00101", -- 01 adi r0, r0, #5
      b"0000000000000101_00000_00000_00000", -- 02 add r0, r0, r0
      b"0000000000001010_00001_00001_00000", -- 03 dec r1, r1
      b"0000000000000111_00111_00001_00101", -- 04 bne -3, r1
      b"0000000000000010_00000_00000_00000", -- 05 st [r0], r0
      b"0000000000000001_00001_00000_00000", -- 06 ld r1, [r0]
```

```
      b"0000000000000011_00001_00001_00000", -- 07 inc r1, r1
      b"0000000000000100_00001_00001_00000", -- 08 not r1, r1
      b"0000000000001001_00001_00000_00001", -- 09 sr r1, r1
      b"0000000000000110_00111_00000_00110", -- 0a b -2
      x"00000000", -- 0b
      x"00000000", -- 0c
      x"00000000", -- 0d
      x"00000000", -- 0e
      x"00000000", -- 0f

        -- module 01
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 02
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 03
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 04
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 05
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 06
        X"00000000", X"00000000", X"00000000", X"00000000",
```

```
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 07
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 08
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 09
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 0A
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 0B
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 0C
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 0D
        X"00000000", X"00000000", X"00000000", X"00000000",
```

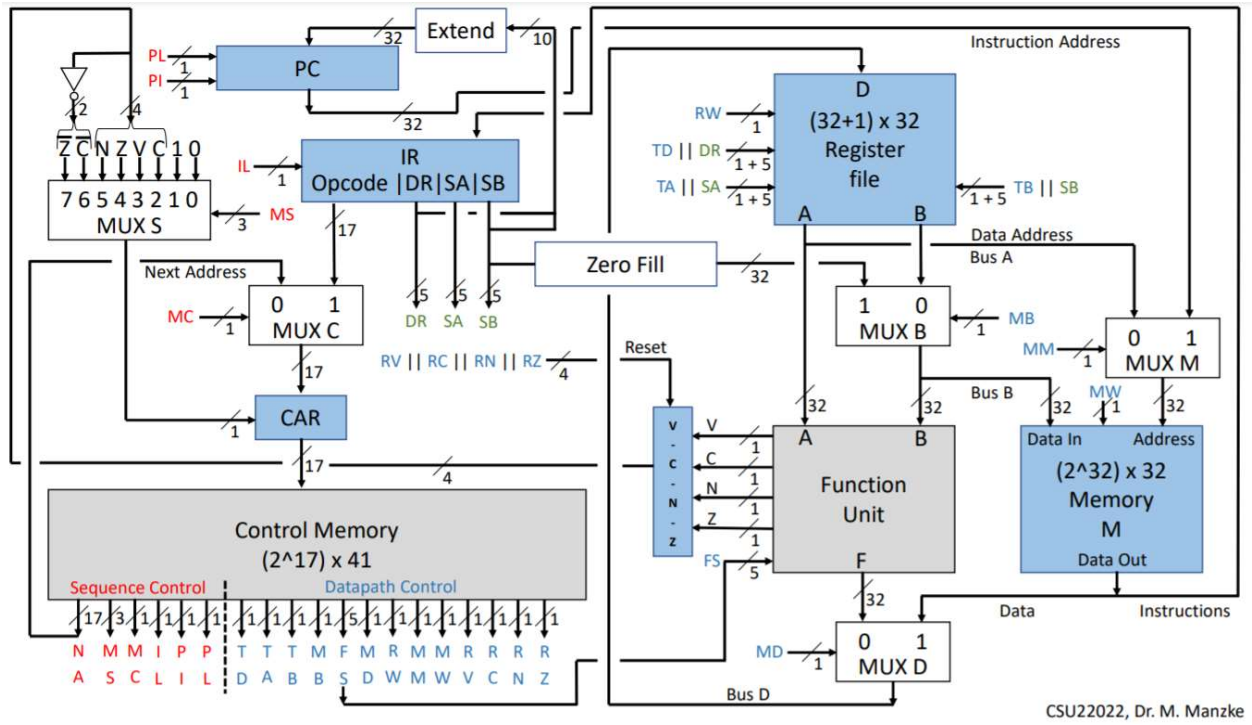
```
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 0E
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 0F
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 10
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 11
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 12
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 13
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 14
        X"00000000", X"00000000", X"00000000", X"00000000",
```

```
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 15
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 16
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 17
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 18
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 19
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 1A
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 1B
        X"00000000", X"00000000", X"00000000", X"00000000",
```

```
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 1C
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 1D
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 1E
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",

        -- module 1F
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000",
        X"00000000", X"00000000", X"00000000", X"00000000"
    );

    variable addr : integer range 0 to 511;

    begin
        addr:=conv_integer(unsigned(address(14 downto 0)));
        read_data <= data_mem(addr) after 1ns;
        if (rising_edge(clk)) then
            if MemWrite ='1' then
                data_mem(addr):= write_data;
                end if;
            else
                read_data <= data_mem(addr) after 1 ns;
        end if;
    end process;
```

```
end Behavioral;
```

Micro Programme Controller :




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MicroprogrammeController is
    Port(   IR : in STD_LOGIC_VECTOR(31 downto 0);
            status_bits : in STD_LOGIC_VECTOR(3 downto 0);
            reset_mpc : in STD_LOGIC;
            clk_mpc : in STD_LOGIC;
            control_word_mpc : out STD_LOGIC_VECTOR(23 downto 0);
            PC_out : out STD_LOGIC_VECTOR(31 downto 0);
            TD_mpc, TA_mpc, TB_mpc, MW_mpc : out STD_LOGIC
        );
end MicroprogrammeController;

architecture Behavioral of MicroprogrammeController is
```

```
    component ControlMemory is
        Port (  FL, RZ, RN, RC, RV, MW, MM, RW, MD, MB, TB, TA, TD, PL, PI,
IL, MC: out std_logic;
                FS : out std_logic_vector(4 downto 0);
                MS : out std_logic_vector(2 downto 0);
                NA : out std_logic_vector(16 downto 0);
                IN_CAR : in std_logic_vector(16 downto 0));
    end component;

    component MUX_2_17Bit is
    Port(   In0_NA, In1_opcode : in STD_LOGIC_VECTOR(16 downto 0);
            S_mc : in STD_LOGIC;
            out_car : out STD_LOGIC_VECTOR(16 downto 0)
        );
    end component;

    COMPONENT MUX8_1Bit
    Port (  In_zero, In_one, In_n, In_z, In_c, In_v, In_not_c, In_not_z :
in STD_LOGIC;
            S : in  STD_LOGIC_VECTOR(2 downto 0);
            Z : out  STD_LOGIC);
    END COMPONENT;

    component CAR is
        Port(   CAR_in : in STD_LOGIC_VECTOR(16 downto 0);
                MUX_S, reset : in STD_LOGIC;
                CAR_out : out STD_LOGIC_VECTOR(16 downto 0)
            );
    end component;

    component InstructionRegister is
        Port ( INSTR : in  STD_LOGIC_VECTOR(31 downto 0);
               IL : in  STD_LOGIC;
               OPCODE : out  STD_LOGIC_VECTOR(16 downto 0);
               DR : out  STD_LOGIC_VECTOR(4 downto 0);
               SA : out  STD_LOGIC_VECTOR(4 downto 0);
               SB : out  STD_LOGIC_VECTOR(4 downto 0));
    end component;

    component ProgrammeCounter is
        Port(   PC_module_in : in STD_LOGIC_VECTOR(31 downto 0);
                PL_module_in, PI_module_in, reset, clock : in STD_LOGIC;
```

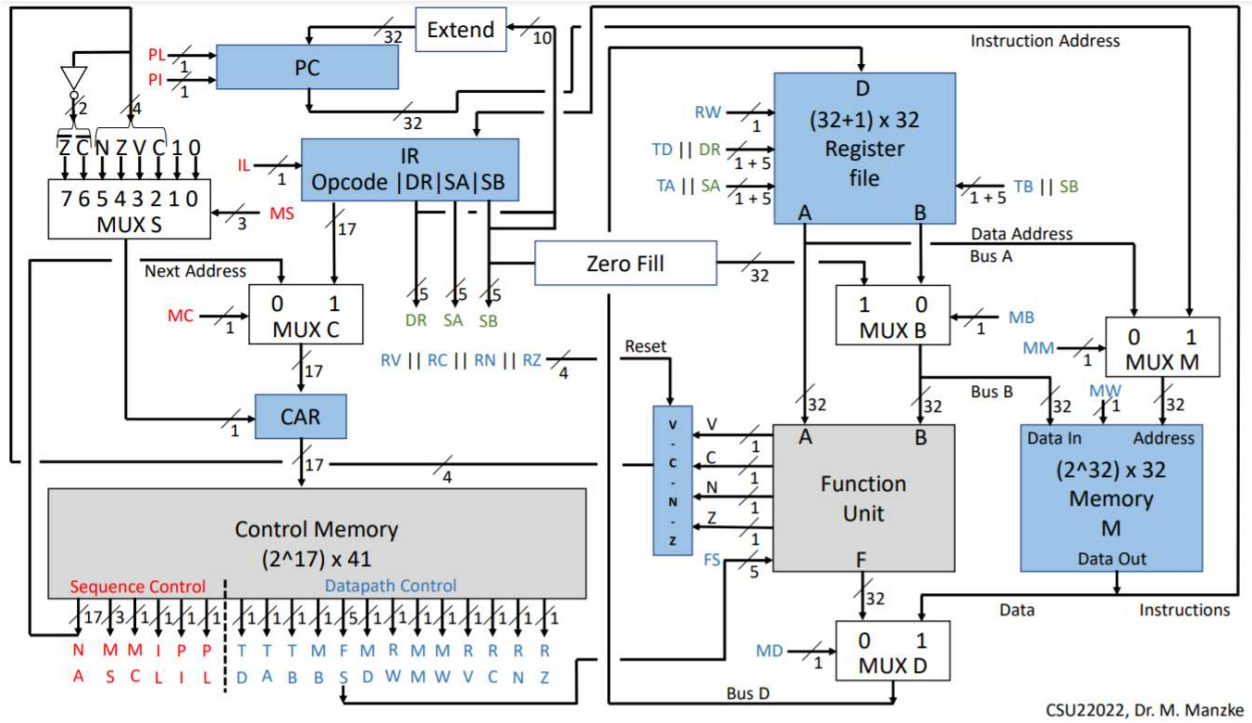
```
                PC_module_out : out STD_LOGIC_VECTOR(31 downto 0)
            );
    end component;

    component ExtendedProgrammeCounter is
    Port( SR_SB : in STD_LOGIC_VECTOR(9 downto 0);
          ExtendedProgrammeCounter : out STD_LOGIC_VECTOR(31 downto 0)
        );
    end component;

    --signalling
    signal control_word_sig : STD_LOGIC_VECTOR(23 downto 0):= (others =>
'0');

    -- CM and MUX-C
    signal na_sig : STD_LOGIC_VECTOR(16 downto 0):= (others => '0');

    --CM and CAR
    signal car_out_sig : STD_LOGIC_VECTOR(16 downto 0):= (others => '0');

    --MUX C
    signal opcode_sig : STD_LOGIC_VECTOR(16 downto 0):= (others => '0');

    --MUX C and CAR
    signal  out_car_sig : STD_LOGIC_VECTOR(16 downto 0):= (others => '0');

    signal out_s_car_sig, mc_sig : STD_LOGIC :='0';

    -- CM and IR
    signal il_sig : STD_LOGIC :='0';
    -- CM and PC
    signal pl_sig, pi_sig : STD_LOGIC := '0';

    -- MS in CM and in MUX S (MUX 8)
    signal ms_cm_sig: STD_LOGIC_VECTOR(2 downto 0):= (others => '0');

    -- IR
    signal sa_sig, sb_sig, dr_sig : STD_LOGIC_VECTOR(4 downto 0):= (others
=> '0');

    -- Extended PC
    signal extend_in : STD_LOGIC_VECTOR(9 downto 0):= (others => '0');
```

```
    signal extend_out : STD_LOGIC_VECTOR(31 downto 0):= (others => '0');

    -- MUX 8
    signal not_c, not_z : std_logic :='0';

begin
    control_memory_mpc : ControlMemory PORT MAP(
        IN_CAR => car_out_sig,
        MW => mw_mpc,
        MM => control_word_sig(0),
        RW => control_word_sig(1),
        MD => control_word_sig(2),
        MB => control_word_sig(8),
        TB => tb_mpc,
        TA => ta_mpc,
        TD => td_mpc,
        PL => pl_sig,
        PI => pi_sig,
        IL => il_sig,
        MC => mc_sig,
        FS => control_word_sig(7 downto 3),
        MS => ms_cm_sig,
        NA => na_sig
    );

    mux2_17_mpc : MUX_2_17Bit PORT MAP(
        In0_NA => na_sig,
        In1_opcode => opcode_sig,
        S_mc => mc_sig,
        out_car => out_car_sig
    );

    not_z <= NOT status_bits(0);
    not_c <= NOT status_bits(2);

    mux8_1_mpc : MUX8_1Bit PORT MAP(
        In_zero => '0',
        In_one => '1',
        In_z => status_bits(0),
        In_n => status_bits(1),
        In_c => status_bits(2),
        In_v => status_bits(3),
```

```
        In_not_z => not_z,
        In_not_c => not_c,
        S => ms_cm_sig,
        Z => out_s_car_sig
    );

    car_mpc : CAR PORT MAP(
        CAR_in => out_car_sig,
        MUX_S => out_s_car_sig,
        reset => reset_mpc,
        CAR_out => car_out_sig
    );

    IR_mpc : InstructionRegister PORT MAP(
        INSTR => IR,
        IL => il_sig,
        Opcode => opcode_sig(16 downto 0),
        DR => dr_sig,
        SA => sa_sig,
        SB => sb_sig
    );

    extended_pc_mpc : ExtendedProgrammeCounter PORT MAP(
        SR_SB => extend_in,
        ExtendedProgrammeCounter => extend_out
    );

    pc_mpc : ProgrammeCounter PORT MAP(
        PC_module_in => extend_out,
        PL_module_in => pl_sig,
        PI_module_in => pi_sig,
        reset => reset_mpc,
        clock => clk_mpc,
        PC_module_out => pc_out
    );

    extend_in(9 downto 5) <= dr_sig;
    extend_in(4 downto 0) <= sb_sig;

    control_word_sig(23 downto 19) <= dr_sig;
    control_word_sig(18 downto 14) <= sa_sig;
    control_word_sig(13 downto 9) <= sb_sig;
```

```
    control_word_mpc <= control_word_sig;

end Behavioral;
```

Processor:




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Processor is
    Port( Clk, reset: in STD_LOGIC
);
end Processor;

architecture Behavioral of Processor is

    component Datapath_32Bit is
    Port ( CONTROL_WORD : in  STD_LOGIC_VECTOR(23 downto 0);
           CONST_IN : in  STD_LOGIC_VECTOR(31 downto 0);
```

```
            DATA_IN : in  STD_LOGIC_VECTOR(31 downto 0);
            CLK : in  STD_LOGIC;
            TD, TA, TB : in STD_LOGIC;
            ADDR_OUT : out  STD_LOGIC_VECTOR(31 downto 0);
            DATA_OUT : out  STD_LOGIC_VECTOR(31 downto 0);
            status_out : out STD_LOGIC_VECTOR(3 downto 0)
            );
    end component;

    component MicroprogrammeController is
    Port(   IR : in STD_LOGIC_VECTOR(31 downto 0);
            status_bits : in STD_LOGIC_VECTOR(3 downto 0);
            reset_mpc : in STD_LOGIC;
            clk_mpc : in STD_LOGIC;
            control_word_mpc : out STD_LOGIC_VECTOR(23 downto 0);
            PC_out : out STD_LOGIC_VECTOR(31 downto 0);
            TD_mpc, TA_mpc, TB_mpc, MW_mpc : out STD_LOGIC
        );
    end component;


    component memory is -- use unsigned for memory address
    Port ( address : in std_logic_vector(31 downto 0);
            write_data : in std_logic_vector(31 downto 0);
            MemWrite : in std_logic;
            clk : in std_logic;
            read_data : out std_logic_vector(31 downto 0)
            );
    end component;

    signal mm_read_data, mpc_pc_out, dp_data_out, dp_address_out :
STD_LOGIC_VECTOR(31 downto 0):= (others => '0');
    signal mpc_control_word : STD_LOGIC_VECTOR(23 downto 0):= (others =>
'0');
    signal dp_status_out : STD_LOGIC_VECTOR(3 downto 0):= (others => '0');
    signal mpc_TD, mpc_TA, mpc_TB, mpc_MW, mpc_MR : STD_LOGIC := '0';

begin

        mem_module : memory PORT MAP(
        address => dp_address_out,
        write_data => dp_data_out,
```

```
        MemWrite => mpc_MW,
        clk => clk,
        read_data => mm_read_data);


        micro_pc : MicroprogrammeController PORT MAP(
        IR => mm_read_data,
        status_bits => dp_status_out,
        reset_mpc => reset,
        clk_mpc => clk,
        control_word_mpc => mpc_control_word,
        PC_out => mpc_pc_out,
        TD_mpc => mpc_TD,
        TA_mpc => mpc_TA,
        TB_mpc => mpc_TB,
        MW_mpc => mpc_MW);

        datapath : Datapath_32Bit PORT MAP(
        DATA_IN => mm_read_data,
        CONST_IN=> mpc_pc_out,
        CONTROL_WORD =>  mpc_control_word,
        CLK => Clk,
        TD => mpc_TD,
        TA => mpc_TA,
        TB => mpc_TB,
        DATA_OUT => dp_data_out,
        ADDR_OUT => dp_address_out,
        status_out => dp_status_out);

end Behavioral;
```

Test Benches

Register 32 Bit:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY reg32_TB IS
END reg32_TB;

ARCHITECTURE behavior OF reg32_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT reg32
    PORT(   D : in std_logic_vector(31 downto 0);
            load : in std_logic_vector(1 downto 0);
            Clk : in std_logic;
            Q : out std_logic_vector(31 downto 0));
    END COMPONENT;


   --Inputs
   signal D : std_logic_vector(31 downto 0):= (others => '0');
   signal load : std_logic_vector(1 downto 0):= (others => '0');
   signal Clk : std_logic := '0';


      --Outputs
   signal Q : std_logic_vector(31 downto 0);

   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

   constant Clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
uut: reg32 PORT MAP (
   D => D,
   load => load,
```

```
    Clk => Clk,
    Q => Q
         );


    Clk <= not Clk after 5ns;
    stim_proc: process
    begin

         wait for 200 ns;
         load(0) <= '1';
         load(1) <= '1';
         D <= x"ABCDEFFF";


         wait for 200 ns;
         load(0) <= '0';
         load(1) <= '0';
         D <= x"11111111";


         wait for 200 ns;
         load(0) <= '1';
         load(1) <= '1';
         D <= x"19303445";


         wait for 100 ns;
         load(0) <= '1';
         load(1) <= '1';
         D <= x"ffffffff";
    end process;

END;
```

Register File 33 -32 Bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY Register_File_TB IS
END Register_File_TB;

ARCHITECTURE behavior OF Register_File_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Register_File
    PORT(   add_A, add_B, add_D : in std_logic_vector(5 downto 0);
            Clk : in std_logic;
            load : in std_logic;
            D_out_data : in std_logic_vector(31 downto 0);
            A_out, B_out: out std_logic_vector(31 downto 0));

            END COMPONENT;
   --Inputs
    signal add_A, add_B, add_D : std_logic_vector(5 downto 0):= (others =>
'0');
    signal Clk : std_logic := '0';
    signal load :  std_logic := '0';
    signal D_out_data :  std_logic_vector(31 downto 0):= (others => '0');

        --Outputs
    signal A_out, B_out: std_logic_vector(31 downto 0);
--     constant Clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
uut: Register_File PORT MAP (
   add_A => add_A,
   add_B => add_B,
   add_D => add_D,
```

```
   Clk => Clk,
   load => load,
   D_out_data => D_out_data,
   A_out => A_out,
   B_out => B_out);
--

Clk <= not Clk after 5ns;

   stim_proc: process
   begin

    load <= '1';
    add_D <= "000000";
    D_out_data <= x"FFFFFFFF";
    wait for 10ns;

    add_D <= "000001";
    D_out_data <= x"EEEEEEEE";
    wait for 10ns;

    add_D <= "000010";
    D_out_data <= x"DDDDDDDD";
    wait for 10ns;

    add_D <= "000011";
    D_out_data <= x"CCCCCCCC";
    wait for 10ns;

    add_D <= "000100";
    D_out_data <= x"BBBBBBBB";
    wait for 10ns;

    add_D <= "000101";
    D_out_data <= x"AAAAAAAA";
    wait for 10ns;

    add_D <= "000110";
    D_out_data <= x"99999999";
    wait for 10ns;

    add_D <= "000111";
```

```
    D_out_data <= x"88888888";
    wait for 10ns;

    add_D <= "001000";
    D_out_data <= x"77777777";
    wait for 10ns;

    D_out_data <= x"33a33a33";
    add_d <= "100000";
    wait for 100ns;

    load <= '0';
    add_a <= "000000";
    add_b <= "000111";
    wait for 100ns;

    add_a <= "000001";
    add_b <= "000110";
    wait for 100ns;

    add_a <= "000010";
    add_b <= "000101";
    wait for 100ns;

    add_a <= "000011";
    add_b <= "000100";
    wait for 100ns;

    add_a <= "001000";
    add_b <= "000111";
    wait for 100ns;

    add_a <= "100000";
    add_b <= "100000";
    wait for 100ns;


--    wait;

   end process;

END;
```

MUX33_32Bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY MUX_33_32Bit_TB IS
END MUX_33_32Bit_TB;

ARCHITECTURE behavior OF MUX_33_32Bit_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT MUX_33_32Bit
    PORT(  In0, In1, In2, In3, In4, In5, In6, In7, In8, In9, In10, In11,
In12, In13, In14, In15, In16, In17, In18, In19, In20, In21, In22, In23,
In24, In25, In26, In27, In28, In29, In30, In31, In32 : in
std_logic_vector(31 downto 0);
            S0, S1, S2, S3, S4, S5 : in std_logic;
            Z : out std_logic_vector(31 downto 0));
    END COMPONENT;


   --Inputs
   signal In0, In1, In2, In3, In4, In5, In6, In7, In8, In9, In10, In11,
In12, In13, In14, In15, In16, In17, In18, In19, In20, In21, In22, In23,
In24, In25, In26, In27, In28, In29, In30, In31, In32 : std_logic_vector(31
downto 0) := (others => '0');
   signal S0, S1, S2, S3, S4, S5 : STD_LOGIC:= '0';
      --Outputs
   signal Z : std_logic_vector(31 downto 0);

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: MUX_33_32Bit PORT MAP (
          In0 => In0,
          In1 => In1,
          In2 => In2,
```

```
        In3 => In3,
        In4 => In4,
        In5 => In5,
        In6 => In6,
        In7 => In7,
        In8 => In8,
        In9 => In9,
        In10 => In10,
        In11 => In11,
        In12 => In12,
        In13 => In13,
        In14 => In14,
        In15 => In15,
        In16 => In16,
        In17 => In17,
        In18 => In18,
        In19 => In19,
        In20 => In20,
        In21 => In21,
        In22 => In22,
        In23 => In23,
        In24 => In24,
        In25 => In25,
        In26 => In26,
        In27 => In27,
        In28 => In28,
        In29 => In29,
        In30 => In30,
        In31 => In31,
        In32 => In32,

        S0 => S0,
        S1 => S1,
        S2 => S2,
        S3 => S3,
        S4 => S4,
        S5 => S5,
        Z => Z );

-- Stimulus process
stim_proc: process
begin
```

```
--      wait for 10ns;1930 3445
            In0 <= x"11111111";
            In1 <= x"22222222";
            In2 <= x"33333333";
            In3 <= x"44444444";
            In4 <= x"55555555";
            In5 <= x"66666666";
            In6 <= x"77777777";
            In7 <= x"88888888";
            In8 <= x"99999999";
            In9 <= x"10101010";
            In10 <= x"11001100";
            In11 <= x"12121212";
            In12 <= x"13131313";
            In13 <= x"14141414";
            In14 <= x"15151515";
            In15 <= x"16161616";
            In16 <= x"17171717";
            In17 <= x"18181818";
            In18 <= x"19191919";
            In19 <= x"20202020";
            In20 <= x"21212121";
            In21 <= x"22002200";
            In22 <= x"23232323";
            In23 <= x"24242424";
            In24 <= x"25252525";
            In25 <= x"26262626";
            In26 <= x"27272727";
            In27 <= x"28282828";
            In28 <= x"29292929";
            In29 <= x"30303030";
            In30 <= x"31313131";
            In31 <= x"32323232";
            In32 <= x"33003300";

--      - 000000
          wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
        S4 <= '0';
```

```
        S5 <= '0';

--       - 000010
        wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 000100
        wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '1';
        S4 <= '0';
        S5 <= '0';

--       - 000110
        wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 001000
        wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '1';
        S3 <= '0';
        S4 <= '0';
        S5 <= '0';

 --      - 001010
        wait for 29ns;
        S0 <= '0';
        S1 <= '0';
```

```
        S2 <= '1';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 001100
        wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '1';
        S3 <= '1';
        S4 <= '0';
        S5 <= '0';

--       - 001110
         wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '1';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 010000
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '0';
        S3 <= '0';
        S4 <= '0';
        S5 <= '0';

--       - 010010
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '0';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 010100
```

```
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '0';
        S3 <= '1';
        S4 <= '0';
        S5 <= '0';

--       - 010110
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '0';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 011000
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '1';
        S3 <= '0';
        S4 <= '0';
        S5 <= '0';

--       - 011010
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '1';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 011100
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '1';
        S3 <= '1';
        S4 <= '0';
```

```
        S5 <= '0';


 --      - 011110
        wait for 29ns;
        S0 <= '0';
        S1 <= '1';
        S2 <= '1';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 100000
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
        S4 <= '0';
        S5 <= '0';

--      - 100010
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 100100
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '0';
        S3 <= '1';
        S4 <= '0';
        S5 <= '0';

--       - 100110
        wait for 29ns;
        S0 <= '1';
```

```
        S1 <= '0';
        S2 <= '0';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 101000
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '1';
        S3 <= '0';
        S4 <= '0';
        S5 <= '0';

--      - 101010
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '1';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 101100
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '1';
        S3 <= '1';
        S4 <= '0';

--        - 101110
        wait for 29ns;
        S0 <= '1';
        S1 <= '0';
        S2 <= '1';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 110000
```

```
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '0';
        S3 <= '0';
        S4 <= '0';
        S5 <= '0';

 --     - 110010
         wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '0';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 110100
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '0';
        S3<= '1';
        S4 <= '0';
        S5 <= '0';

--      - 110110
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '0';
        S3 <= '1';
        S4 <= '1';
        S5 <= '0';

--       - 111000
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '1';
        S3 <= '0';
        S4 <= '0';
```

```
        S5 <= '0';

--       - 111010
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '1';
        S3 <= '0';
        S4 <= '1';
        S5 <= '0';

--       - 111100
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '1';
        S3 <= '1';
        S4 <= '0';
        S5 <= '0';

--       - 111110
        wait for 29ns;
        S0 <= '1';
        S1 <= '1';
        S2 <= '1';
        S3 <= '1';
        S4 <= '1';
          S5 <= '0';

--       - 000001
          wait for 29ns;
        S0 <= '0';
        S1 <= '0';
        S2 <= '0';
        S3 <= '0';
        S4 <= '0';
        S5 <= '1';

    end process;

END;
```

Decoder_6to33:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY decoder_6to33_tb IS
END decoder_6to33_tb;

ARCHITECTURE behavior OF decoder_6to33_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT decoder_6to33
    PORT(   A0 : in std_logic;
            A1 : in std_logic;
            A2 : in std_logic;
            A3 : in std_logic;
            A4 : in std_logic;
            A5 : in std_logic;

            Q0 : out std_logic;
            Q1 : out std_logic;
            Q2 : out std_logic;
            Q3 : out std_logic;
            Q4 : out std_logic;
            Q5 : out std_logic;
            Q6 : out std_logic;
            Q7 : out std_logic;
            Q8 : out std_logic;
            Q9 : out std_logic;
            Q10 : out std_logic;
            Q11 : out std_logic;
            Q12 : out std_logic;
            Q13 : out std_logic;
            Q14 : out std_logic;
            Q15 : out std_logic;
            Q16 : out std_logic;
            Q17 : out std_logic;
            Q18 : out std_logic;
            Q19 : out std_logic;
            Q20 : out std_logic;
            Q21 : out std_logic;
```

```
            Q22 : out std_logic;
            Q23 : out std_logic;
            Q24 : out std_logic;
            Q25 : out std_logic;
            Q26 : out std_logic;
            Q27 : out std_logic;
            Q28 : out std_logic;
            Q29 : out std_logic;
            Q30 : out std_logic;
            Q31 : out std_logic;
            Q32 : out std_logic);
    END COMPONENT;


   --Inputs
   signal A0 : std_logic;
   signal A1 : std_logic;
   signal A2 : std_logic;
   signal A3 : std_logic;
   signal A4 : std_logic;
   signal A5 : std_logic;

      --Outputs
   signal Q0 : std_logic ;
   signal Q1 : std_logic;
   signal Q2 : std_logic;
   signal Q3 : std_logic;
   signal Q4 : std_logic;
   signal Q5 : std_logic;
   signal Q6 : std_logic;
   signal Q7 : std_logic;
   signal Q8 : std_logic;
   signal Q9 : std_logic;
   signal Q10 : std_logic;
   signal Q11 : std_logic;
   signal Q12 : std_logic;
   signal Q13 : std_logic;
   signal Q14 : std_logic;
   signal Q15 : std_logic;
   signal Q16 : std_logic;
   signal Q17 : std_logic;
   signal Q18 : std_logic;
```

```
   signal Q19 : std_logic;
   signal Q20 : std_logic;
   signal Q21 : std_logic;
   signal Q22 : std_logic;
   signal Q23 : std_logic;
   signal Q24 : std_logic;
   signal Q25 : std_logic;
   signal Q26 : std_logic;
   signal Q27 : std_logic;
   signal Q28 : std_logic;
   signal Q29 : std_logic;
   signal Q30 : std_logic;
   signal Q31 : std_logic;
   signal Q32 : std_logic;

   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

--   constant Clk_period : time := 5 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
uut: decoder_6to33 PORT MAP (
   A0 => A0,
   A1 => A1,
   A2 => A2,
   A3 => A3,
   A4 => A4,
   A5 => A5,

   Q0 => Q0,
   Q1 => Q1,
   Q2 => Q2,
   Q3 => Q3,
   Q4 => Q4,
   Q5 => Q5,
   Q6 => Q6,
   Q7 => Q7,
   Q8 => Q8,
   Q9 => Q9,
   Q10 => Q10,
```

```
   Q11 => Q11,
   Q12 => Q12,
   Q13 => Q13,
   Q14 => Q14,
   Q15 => Q15,
   Q16 => Q16,
   Q17 => Q17,
   Q18 => Q18,
   Q19 => Q19,
   Q20 => Q20,
   Q21 => Q21,
   Q22 => Q22,
   Q23 => Q23,
   Q24 => Q24,
   Q25 => Q25,
   Q26 => Q26,
   Q27 => Q27,
   Q28 => Q28,
   Q29 => Q29,
   Q30 => Q30,
   Q31 => Q31,
   Q32 => Q32
        );

   stim_proc: process
   begin

--     0 - 000000
       wait for 29 ns;
       A0 <= '0';
       A1 <= '0';
       A2 <= '0';
       A3 <= '0';
       A4 <= '0';
       A5 <= '0';

--     1 - 000010
       wait for 29 ns;
       A0 <= '0';
       A1 <= '0';
       A2 <= '0';
       A3 <= '0';
```

```
        A4 <= '1';
        A5 <= '0';

--      2 - 000100
        wait for 29 ns;
        A0 <= '0';
        A1 <= '0';
        A2 <= '0';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

--      3 - 000110
        wait for 29 ns;
        A0 <= '0';
        A1 <= '0';
        A2 <= '0';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--      4 - 001000
        wait for 29 ns;
        A0 <= '0';
        A1 <= '0';
        A2 <= '1';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';

 --     5 - 001010
        wait for 29 ns;
        A0 <= '0';
        A1 <= '0';
        A2 <= '1';
        A3 <= '0';
        A4 <= '1';
        A5 <= '0';

--      6 - 001100
        wait for 29 ns;
        A0 <= '0';
```

```
        A1 <= '0';
        A2 <= '1';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

--       7 - 001110
         wait for 29 ns;
        A0 <= '0';
        A1 <= '0';
        A2 <= '1';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--      8 - 010000
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '0';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';

--      9 - 010010
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '0';
        A3 <= '0';
        A4 <= '1';
        A5 <= '0';

--      10 - 010100
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '0';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';
```

```
--      11- 010110
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '0';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--      12 - 011000
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '1';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';

--      13- 011010
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '1';
        A3 <= '0';
        A4 <= '1';
        A5 <= '0';

--      14 - 011100
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '1';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

 --     15 - 011110
        wait for 29 ns;
        A0 <= '0';
        A1 <= '1';
        A2 <= '1';
        A3 <= '1';
```

```
        A4 <= '1';
        A5 <= '0';

--      16 - 100000
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '0';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';

--       17 - 100010
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '0';
        A3 <= '0';
        A4 <= '1';
        A5 <= '0';

--       18 - 100100
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '0';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

--       19 - 100110
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '0';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--       20 - 101000
        wait for 29 ns;
        A0 <= '1';
```

```
        A1 <= '0';
        A2 <= '1';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';

--      21 - 101010
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '1';
        A3 <= '0';
        A4 <= '1';
        A5 <= '0';

--       22 - 101100
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '1';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

--       23 - 101110
        wait for 29 ns;
        A0 <= '1';
        A1 <= '0';
        A2 <= '1';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--      24 - 110000
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '0';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';
```

```
 --      25 - 110010
         wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '0';
        A3 <= '0';
        A4 <= '1';
        A5 <= '0';

--      26 - 110100
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '0';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

--      27 - 110110
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '0';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--      28 - 111000
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '1';
        A3 <= '0';
        A4 <= '0';
        A5 <= '0';

--      29 - 111010
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '1';
        A3 <= '0';
```

```
        A4 <= '1';
        A5 <= '0';

--       30 - 111100
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '1';
        A3 <= '1';
        A4 <= '0';
        A5 <= '0';

--       31 - 111110
        wait for 29 ns;
        A0 <= '1';
        A1 <= '1';
        A2 <= '1';
        A3 <= '1';
        A4 <= '1';
        A5 <= '0';

--       32 - 000001
        wait for 29 ns;
        A0 <= '0';
        A1 <= '0';
        A2 <= '0';
        A3 <= '0';
        A4 <= '0';
        A5 <= '1';

--        wait;
    end process;

END;
```

Ripple Adder:

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

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity RippleAdder_32Bit_TB is
end RippleAdder_32Bit_TB;

architecture Behavioral of RippleAdder_32Bit_TB is

COMPONENT Ripple_Adder
    PORT(
          A, B : in STD_LOGIC_VECTOR(31 downto 0);
        Cin : in STD_LOGIC;
        Cout, V_out : out STD_LOGIC;
        G_out : out STD_LOGIC_VECTOR(31 downto 0)
    );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(31 downto 0) := (others => '0');
   signal B : std_logic_vector(31 downto 0) := (others => '0');
   signal Cin : std_logic := '0';

    --Outputs
    signal V_out : std_logic;
   signal Cout : std_logic;
   signal G_out : std_logic_vector(31 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Ripple_Adder PORT MAP (
          A => A,
          B => B,
          Cin => Cin,
          V_out => V_out,
          Cout => Cout,
          G_out => G_out
        );

   -- Stimulus process
   stim_proc: process
```

```
   begin
      A <= x"11112222";
        B <= x"22221111";
        Cin <= '0';
        wait for 100ns;

        A <= x"33334444";
        B <= x"44443333";
        Cin <= '1';
        wait for 100ns;

        A <= x"ffffffff";
        B <= x"ffffffff";
        Cin <= '0';
        wait for 100ns;

        A <= x"01010101";
        B <= x"10101010";
        Cin <= '1';
        wait for 100ns;
   end process;

END;
```

Full Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FullAdder_TB is
end FullAdder_TB;

architecture Behavioral of FullAdder_TB is

    COMPONENT Full_Adder
    PORT(
         X : IN  std_logic;
         Y : IN  std_logic;
         Cin : IN  std_logic;
```

```
         Sum : OUT  std_logic;
         Cout : OUT  std_logic
        );
    END COMPONENT;

   -- Input signals
   signal X : std_logic := '0';
   signal Y : std_logic := '0';
   signal Cin : std_logic := '0';

    -- Outputs signals
   signal Sum : std_logic;
   signal Cout : std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Full_Adder PORT MAP (
          X => X,
          Y => Y,
          Cin => Cin,
          Sum => Sum,
          Cout => Cout
        );

  -- Stimulus process
   stim_proc: process
   begin
        Cin <= '0';
        Y <= '0';
        X <= '0';
        wait for 100ns;

        Cin <= '0';
        Y <= '0';
        X <= '1';
        wait for 100ns;

        Cin <= '0';
        Y <= '1';
        X <= '0';
        wait for 100ns;
```

```
        Cin <= '0';
        Y <= '1';
        X <= '1';
        wait for 100ns;

        Cin <= '1';
        Y <= '0';
        X <= '0';
        wait for 100ns;

        Cin <= '1';
        Y <= '0';
        X <= '1';
        wait for 100ns;

        Cin <= '1';
        Y <= '1';
        X <= '0';
        wait for 100ns;

        Cin <= '1';
        Y <= '1';
        X <= '1';
        wait for 100ns;

   end process;

END;
```

Logic Unit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LogicUnit_TB is
end LogicUnit_TB;

architecture Behavioral of LogicUnit_TB is
```

```
COMPONENT Logic_Unit
    PORT(
         A : IN  std_logic_vector(31 downto 0);
         B : IN  std_logic_vector(31 downto 0);
         S : IN  std_logic_vector(1 downto 0);
         G : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(31 downto 0) := (others => '0');
   signal B : std_logic_vector(31 downto 0) := (others => '0');
   signal S : std_logic_vector(1 downto 0) := (others => '0');

    --Outputs
   signal G : std_logic_vector(31 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Logic_Unit PORT MAP (
          A => A,
          B => B,
          S => S,
          G => G
        );

-- Stimulus process
   stim_proc: process
   begin
        A <= x"11111111";
        B <= x"00000000";

        S(0) <= '0';
        S(1) <= '0';
        wait for 200ns;

        S(0) <= '0';
        S(1) <= '1';
        wait for 200ns;
```

```
        S(0) <= '1';
        S(1) <= '0';
        wait for 200ns;

        S(0) <= '1';
        S(1) <= '1';
        wait for 200ns;
   end process;

END;
```

MUX 3 1 Bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


ENTITY MUX3_1Bit_TB IS
END MUX3_1Bit_TB;

ARCHITECTURE behavior OF MUX3_1Bit_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT MUX_3_1Bit
    Port(
        In0, In1, In2 : in STD_LOGIC;
        S0, S1 : in STD_LOGIC;
        Z : out STD_LOGIC
    );
    END COMPONENT;


   --Inputs
   signal S0, S1 : std_logic := '0';
   signal In0, In1, In2 : std_logic := '0';

      --Outputs
   signal Z : std_logic := '0';

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: MUX_3_1Bit PORT MAP (
          In0 => In0,
          In1 => In1,
          In2 => In2,
          S0 => S0,
          S1 => S1,
          Z => Z
        );

   -- Stimulus process
```

```
   stim_proc: process
   begin
--      wait for 10ns;
            In0 <= '1';
            In1 <= '0';
            In2 <= '1';

            wait for 10ns;
            S0 <= '0';
            S1 <= '0';

            wait for 10ns;
            S0 <= '0';
            S1 <= '1';

            wait for 10ns;
            S0 <= '1';
            S1 <= '0';

            wait for 10ns;
            S0 <= '1';
            S1 <= '1';

            wait for 10ns;
            In0 <= '0';
            In1 <= '1';
            In2 <= '0';

            wait for 10ns;
            S0 <= '0';
            S1 <= '0';

            wait for 10ns;
            S0 <= '0';
            S1 <= '1';

            wait for 10ns;
            S0 <= '1';
            S1 <= '0';

            wait for 10ns;
            S0 <= '1';
```

```
            S1 <= '1';

   end process;

END;
```

Function Unit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Function_Unit_TB IS
END Function_Unit_TB;

ARCHITECTURE behavior OF Function_Unit_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Function_Unit
    PORT(
         A : IN  std_logic_vector(31 downto 0);
         B : IN  std_logic_vector(31 downto 0);
         FSel : IN  std_logic_vector(4 downto 0);
         C : OUT  std_logic;
         V : OUT  std_logic;
         N : OUT  std_logic;
         Z : OUT  std_logic;
         F : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal A : std_logic_vector(31 downto 0) := (others => '0');
   signal B : std_logic_vector(31 downto 0) := (others => '0');
   signal FSel : std_logic_vector(4 downto 0) := (others => '0');

    --Outputs
```

```
   signal C : std_logic;
   signal V : std_logic;
   signal N : std_logic;
   signal Z : std_logic;
   signal F : std_logic_vector(31 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Function_Unit PORT MAP (
          A => A,
          B => B,
          FSel => FSel,
          C => C,
          V => V,
          N => N,
          Z => Z,
          F => F
        );

   -- Stimulus process
   stim_proc: process
   begin
        -- TRANSFER A
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00000";
        wait for 65ns; -- F = 0000 0001

        -- INCREMENT
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00001";
        wait for 65ns; -- F = 0000 0010

        -- ADD
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00010";
        wait for 65ns; -- F = 0000 0011

        -- ADD WITH CARRY
```

```
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00011";
        wait for 65ns; -- F = 0000 0100

        -- ADD WITH NOT B
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00100";
        wait for 65ns; -- F = 1111 1110

        -- SUBTRACT
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00101";
        wait for 65ns; -- F = 1111 1111

        -- DECREMENT
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00110";
        wait for 65ns; -- F = 0000 0000

        -- TRANSFER A
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "00111";
        wait for 65ns; -- F = 0000 0001

        -- AND

        A <= x"00000001";
        B <= x"00000002";
        FSel <= "01000";
        wait for 65ns; -- F = 0000 0000

        -- OR
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "01010";
        wait for 65ns; -- F = 0000 0011
```

```
        -- XOR
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "01100";
        wait for 65ns; -- F = 0000 0011

        -- NOT
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "01110";
        wait for 65ns; -- F = 1111 1110

        -- TRANSFER B
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "10000";
        wait for 65ns; -- F = 0000 0010

        -- SHIFT RIGHT (also displays carry)
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "10100";
        wait for 65ns; -- F = 0000 0001

        -- SHIFT LEFT (also displays carry)
        A <= x"00000001";
        B <= x"00000002";
        FSel <= "11000";
        wait for 65ns; -- F = 0000 0100
    end process;

END;
```

Datapath:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Datapath_32B_TB is
end Datapath_32B_TB;

architecture Behavioral of Datapath_32B_TB is

COMPONENT Datapath_32Bit
   Port (
          CONTROL_WORD : in  STD_LOGIC_VECTOR(23 downto 0);
          CONST_IN : in  STD_LOGIC_VECTOR(31 downto 0);
          DATA_IN : in  STD_LOGIC_VECTOR(31 downto 0);
          CLK : in  STD_LOGIC;
          TD, TA, TB : in STD_LOGIC;
          ADDR_OUT : out  STD_LOGIC_VECTOR(31 downto 0);
          DATA_OUT : out  STD_LOGIC_VECTOR(31 downto 0);
          status_out : out STD_LOGIC_VECTOR(3 downto 0)
          );
    END COMPONENT;


   --Inputs
   signal CONST_IN : std_logic_vector(31 downto 0) := (others => '0');
   signal DATA_IN : std_logic_vector(31 downto 0) := (others => '0');
   signal CLK : std_logic := '0';
   signal CONTROL_WORD : STD_LOGIC_VECTOR(23 downto 0) := (others => '0');
   -- |2   1|1   1|1   0|0|0|0|0|0|0|0|0|0|
   -- |3   9|8   3|2   9|8|7       3|2|1|0|
   -- |  D  |  A  |  B  |M|  F SEL  |M|L|M|
   -- |DEST | ADD | ADD |B|  F SEL  |D|D|M|

    --Outputs
   signal ADDR_OUT : std_logic_vector(31 downto 0);
   signal DATA_OUT : std_logic_vector(31 downto 0);
   signal status_out : std_logic_vector(3 downto 0);
   signal TD, TA, TB : std_logic := '0';
   signal C, V, N, Z : std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: Datapath_32Bit PORT MAP (
          CONTROL_WORD => CONTROL_WORD,
          CONST_IN => CONST_IN,
```

```
          DATA_IN => DATA_IN,
          CLK => CLK,
          TD => TD,
          TA => TA,
          TB => TB,
          ADDR_OUT => ADDR_OUT,
          DATA_OUT => DATA_OUT,
          status_out => status_out
        );



CLK <= not CLK after 5ns;
   -- Stimulus process
   stim_proc: process
   begin

     CONST_IN <= x"00000023";

--      A_add = 00000 , B_add = 00000 , D_add = 00000 MB_Sel =1, MD_sel =1,
Load = 1 F_Sel = 00000
--      R0 = DATA_IN
      DATA_IN <= x"00000001";
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00000_00000_00000_1_00000_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      A_add = 00000 , B_add = 00000 , D_add = 00001 MB_Sel =1, MD_sel =1,
Load = 1 F_Sel = 00001
--      R1 = R0 + 1
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00001_00000_00000_1_00001_1_1_0";
```

```
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      loading R1 to Data_out
--      A_add = 00000 , B_add = 00001 , D_add = 00010 MB_Sel =0, MD_sel =0,
Load = 1 F_Sel = 00010
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00010_00000_00001_0_00010_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;
--      wait;
 --      R1 = DATA_IN = 2
 --      A_add = 00000 , B_add = 00001 , D_add = 00001 MB_Sel =1, MD_sel
=1, Load = 1 F_Sel = 00000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00001_00000_00001_1_00000_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      DATA_IN <= x"00000002";
      wait for 35ns;

     ---- R2 = R0 + R1
--      A_add = 00001 , B_add = 00000 , D_add = 00010 , MB_Sel =0, MD_sel
=0, Load = 1 F_Sel = 00010
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
```

```
      CONTROL_WORD <= b"00010_00001_00000_0_00010_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

 -- loading R2 to  Data_Out
      --      A_add = 00000 , B_add = 00010 , D_add = 00010 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00010_00000_00010_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      ---- R3 = R0 + R1 + 1
--      A_add = 00000 , B_add = 00001 , D_add = 00011 , MB_Sel =1, MD_sel
=1, Load = 1 F_Sel = 00011
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00011_00000_00001_1_00011_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

       -- loading R3 to  Data_Out
      --      A_add = 00000 , B_add = 00011 , D_add = 00011 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
```

```
      CONTROL_WORD <= b"00011_00000_00011_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- R4 = R1 + (R2 Bar)
      --      A_add = 00001 , B_add = 00010 , D_add = 00100 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 00100
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00100_00001_00010_1_00100_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R4 to  Data_Out
      --      A_add = 00000 , B_add = 00100 , D_add = 00100 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00100_00000_00100_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- R5 = R1 + (R2 Bar) + 1
      --      A_add = 00001 , B_add = 00010 , D_add = 00101 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 00101
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
```

```
      CONTROL_WORD <= b"00101_00001_00010_1_00101_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R5 to  Data_Out
      --      A_add = 00000 , B_add = 00101 , D_add = 00101 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00101_00000_00101_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- R6 = R1 - 1
      --      A_add = 00000 , B_add = 00001 , D_add = 00110 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 00110
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00110_00000_00001_1_00110_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R6 to  Data_Out
      --      A_add = 00000 , B_add = 00110 , D_add = 00101 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
```

```
      CONTROL_WORD <= b"00101_00000_00110_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- R7 = R1
      --      A_add = 00000 , B_add = 00001 , D_add = 00111 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 00111
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00111_00000_00001_1_00111_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R7 to  Data_Out
      --      A_add = 00000 , B_add = 00111 , D_add = 00111 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                  -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"00111_00000_00111_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      G = A  B F_Sel=01000 And
    -- R8 = R1 ^ R2
      --      A_add = 00001 , B_add = 00010 , D_add = 01000 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 01000
                  -- |2   1|1   1|1   0|0|0   0|0|0|0|
                  -- |3   9|8   4|3   9|8|7   3|2|1|0|
                  -- |  D   |  A   |  B   |M|F SEL|M|L|M|
```

```
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01101_00000_01101_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R8 to  Data_Out
      --      A_add = 00000 , B_add = 01000 , D_add = 01000 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D  |  A  |  B  |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01000_00000_01000_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      G = A  B F_Sel = 01010 OR
    -- R9 = R1 v R2
      --      A_add = 00001 , B_add = 00010 , D_add = 01001 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 01010
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D  |  A  |  B  |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01001_00001_00010_1_01010_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R9 to  Data_Out
      --      A_add = 00000 , B_add = 01001 , D_add = 01001 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
```

```
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01001_00000_01001_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      G = A xor B F_Sel = 01100 XOR
    -- R10 = R1 xor R2
      --      A_add = 00001 , B_add = 00010 , D_add = 01010 MB_Sel =1,
MD_sel =1, Load = 1 F_Sel = 01100
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01010_00010_00010_1_01100_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R10 to  Data_Out
      --      A_add = 00000 , B_add = 01010 , D_add = 01010 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01010_00000_01010_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      R11 =  - R0
--      A_add = 00000 , B_add = 00001 , D_add = 01011 MB_Sel =0, MD_sel =0,
Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
```

```
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01011_00000_00001_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R11 to  Data_Out
      --      A_add = 00000 , B_add = 01011 , D_add = 01011 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01011_00000_01011_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      R12 =  R1
--      A_add = 00000 , B_add = 00001 , D_add = 01100 MB_Sel =0, MD_sel =0,
Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01100_00000_00001_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R12 to  Data_Out
      --      A_add = 00000 , B_add = 01100 , D_add = 01100 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
```

```
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01100_00000_01100_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      --      R13 =  sr R1
--      A_add = 00000 , B_add = 00001 , D_add = 01101 MB_Sel =1, MD_sel =1,
Load = 1 F_Sel = 10100
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01101_00000_00001_1_10100_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R13 to  Data_Out
      --      A_add = 00000 , B_add = 01101 , D_add = 01101 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D   |  A   |  B   |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01101_00000_01101_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

--      R14 =  sl R1
--      A_add = 00000 , B_add = 00001 , D_add = 01110 MB_Sel =1, MD_sel =1,
Load = 1 F_Sel = 11000
                    -- |2   1|1   1|1   0|0|0   0|0|0|0|
```

```
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D  |  A  |  B  |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01110_00000_00001_1_11000_1_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;

      -- loading R14 to  Data_Out
      --      A_add = 00000 , B_add = 01101 , D_add = 01101 MB_Sel =0,
MD_sel =0, Load = 1 F_Sel = 10000
                    -- |2   1|1   1|1   0|0|0|0   0|0|0|0|
                    -- |3   9|8   4|3   9|8|7   3|2|1|0|
                    -- |  D  |  A  |  B  |M|F SEL|M|L|M|
                    -- |DEST | ADD | ADD |B|F SEL|D|D|M|
      CONTROL_WORD <= b"01101_00000_01101_0_10000_0_1_0";
      Z <= status_out(0);
      N <= status_out(1);
      C <= status_out(2);
      V <= status_out(3);
      wait for 35ns;
      wait;
   end process;

END;
```

Zero Fill:

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

entity ZeroFillTB is
end ZeroFillTB;

architecture Behavioral of ZeroFillTB is

COMPONENT ZeroFill
    PORT(
        SB_in : IN std_logic_vector(4 downto 0);
        ZeroFill_out : OUT std_logic_vector(31 downto 0)
```

```
        );
    END COMPONENT;

    --Inputs
    signal SB_in : std_logic_vector(4 downto 0) := (others => '0');

    --Outputs
    signal ZeroFill_out : std_logic_vector(31 downto 0);

BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: ZeroFill PORT MAP (
    SB_in => SB_in,
    ZeroFill_out => ZeroFill_out
    );
-- Stimulus process
stim_proc: process
    begin
        wait for 200ns;
        SB_in <= "11111";

        wait for 200ns;
        SB_in <= "11100";

        wait for 200ns;
        SB_in <= "11011";

        wait for 200ns;
        SB_in <= "10001";

        wait for 200ns;
        SB_in <= "00000";

        wait for 200ns;
        SB_in <= "10101";
--        wait;
    end process;
END;
```

MUX8 1Bit:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity MUX8_1Bit_TB is
end MUX8_1Bit_TB;

architecture Behavioral of MUX8_1Bit_TB is

COMPONENT MUX8_1Bit
    Port (  In_zero, In_one, In_n, In_z, In_c, In_v, In_not_c, In_not_z :
in STD_LOGIC;
            S : in  STD_LOGIC_VECTOR(2 downto 0);
            Z : out  STD_LOGIC);
    END COMPONENT;


   --Inputs
    signal In_zero : std_logic := '0';
    signal In_one : std_logic := '0';
    signal In_n : std_logic := '0';
    signal In_z : std_logic := '0';
    signal In_c : std_logic := '0';
    signal In_v : std_logic := '0';
    signal In_not_c : std_logic := '0';
    signal In_not_z : std_logic := '0';

    signal S : STD_LOGIC_VECTOR(2 downto 0):= (others => '0');

      --Outputs
   signal Z : std_logic;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: MUX8_1Bit PORT MAP (
            In_zero => In_zero,
            In_one => In_one,
            In_n => In_n,
            In_z => In_z,
            In_c => In_c,
            In_v => In_v,
            In_not_c => In_not_c,
            In_not_z => In_not_z,
```

```
            S => S,
            Z => Z
        );

   -- Stimulus process
   stim_proc: process
   begin
--      wait for 10ns;
            In_one <= '1';
        In_c <= '1';
        In_n <= '1';
        In_v <= '1';
        In_not_z <= '1';


            S(0) <= '0';
            S(1) <= '0';
            S(2) <= '0';

            wait for 200ns;

            In_c <= '0';
            In_z <= '1';
        In_n <= '0';
        In_v <= '0';
        In_not_z <= '0';
        In_not_c <= '1';

            S(0) <= '1';
            S(1) <= '1';
            S(2) <= '0';

            wait for 200ns;

            In_c <= '0';
            In_z <= '1';
        In_n <= '0';
        In_v <= '0';
        In_not_z <= '1';
        In_not_c <= '0';

            S(0) <= '1';
```

```
            S(1) <= '1';
            S(2) <= '1';

            wait for 200ns;

            In_c <= '1';
            In_z <= '0';
        In_n <= '1';
        In_v <= '1';
        In_not_z <= '0';
        In_not_c <= '1';

            S(0) <= '1';
            S(1) <= '1';
            S(2) <= '1';

            wait for 200ns;


   end process;

END;
```

MUX2 17Bit:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY MUX_2_17Bit_TB IS
END MUX_2_17Bit_TB;

ARCHITECTURE behavior OF MUX_2_17Bit_TB IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT MUX_2_17Bit
    PORT(
    In0_NA : IN std_logic_vector(16 downto 0);
    In1_opcode : IN std_logic_vector(16 downto 0);
    S_mc : IN std_logic;
    out_car : OUT std_logic_vector(16 downto 0)
    );
    END COMPONENT;
```

```
    --Inputs
    signal In0_NA : std_logic_vector(16 downto 0) := (others => '0');
    signal In1_opcode : std_logic_vector(16 downto 0) := (others => '0');
    signal S_mc : std_logic := '0';

    --Outputs
    signal out_car : std_logic_vector(16 downto 0);

BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: MUX_2_17Bit PORT MAP (
        In0_NA => In0_NA,
        In1_opcode => In1_opcode,
        S_mc => S_mc,
        out_car => out_car
        );

    -- Stimulus process
    stim_proc: process
    begin
        In0_NA <= "10000000000000001";
        In1_opcode <= "00000011111110000";
        wait for 500ns;
        S_mc <= '1';
--        wait;
    end process;
END;
```

Instruction Register:

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

ENTITY InstructionRegister_TB IS
END InstructionRegister_TB;

ARCHITECTURE behavior OF InstructionRegister_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT InstructionRegister
```

```
    PORT(
         INSTR : IN  std_logic_vector(31 downto 0);
         IL : IN  std_logic;
         OPCODE : OUT  std_logic_vector(16 downto 0);
         DR : OUT  std_logic_vector(4 downto 0);
         SA : OUT  std_logic_vector(4 downto 0);
         SB : OUT  std_logic_vector(4 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal INSTR : std_logic_vector(31 downto 0) := (others => '0');
   signal IL : std_logic := '0';

    --Outputs
   signal OPCODE : std_logic_vector(16 downto 0);
   signal DR : std_logic_vector(4 downto 0);
   signal SA : std_logic_vector(4 downto 0);
   signal SB : std_logic_vector(4 downto 0);


BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: InstructionRegister PORT MAP (
          INSTR => INSTR,
          IL => IL,
          OPCODE => OPCODE,
          DR => DR,
          SA => SA,
          SB => SB
        );


   -- Stimulus process
   stim_proc: process
   begin
        INSTR <= b"0_0000_0000_0000_0000" & b"00000_00000_00000";
        IL <= '0';
        wait for 200ns; -- Output should remain all 0's as IL = 0
```

```
        INSTR <= b"0_1111_0000_1111_0000" & b"10000_00100_00001";
        IL <= '1';
        wait for 200ns; -- Output will be
01111000011110000_10000_00100_00001

        INSTR <= b"0_1111_1111_1111_0000" & b"10001_10100_00011";
        IL <= '0';
        wait for 200ns; -- Output should remain the same as previous as IL
= 0

        INSTR <= b"1_1111_1111_1111_1111" & b"11111_11111_11111";
        IL <= '1';
        wait for 200ns; -- Output should be
1111111111111111_11111_11111_11111
   end process;

END;
```

Program Counter:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ProgrammeCounter_TB IS
END ProgrammeCounter_TB;

ARCHITECTURE behavior OF ProgrammeCounter_TB IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT ProgrammeCounter
    PORT(
        PC_module_in : IN std_logic_vector(31 downto 0);
        PL_module_in : IN std_logic;
        PI_module_in : IN std_logic;
        reset, clock : IN std_logic;
        PC_module_out : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    --Inputs
    signal PC_module_in : std_logic_vector(31 downto 0) := (others => '0');
    signal PL_module_in : std_logic := '0';
```

```
    signal PI_module_in : std_logic := '0';
    signal reset : std_logic := '0';
    signal clock: std_logic := '0';

    --Outputs
    signal PC_module_out : std_logic_vector(31 downto 0);


BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ProgrammeCounter PORT MAP (
            PC_module_in => PC_module_in,
            PL_module_in => PL_module_in,
            PI_module_in => PI_module_in,
            reset => reset,
            clock => clock,
            PC_module_out => PC_module_out
    );

    -- Stimulus process
    clock <= not clock after 25ns;
    stim_proc: process
    begin
        reset <= '1';
        wait for 100ns;

        reset <= '0';
        PL_module_in <= '0';
        PI_module_in <= '1';

        wait for 200ns;
        PC_module_in <= x"00001111";
        PL_module_in <= '1';
        PI_module_in <= '0';

        wait for 100ns;
        PC_module_in <= x"0000ffff";
        PL_module_in <= '1';
        PI_module_in <= '0';

        wait for 100ns;
```

```
        PL_module_in <= '0';
        PI_module_in <= '0';

        wait;
    end process;

END;
```

Extended Program Counter:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ExtendedProgrammeCounter_TB IS
END ExtendedProgrammeCounter_TB;

ARCHITECTURE behavior OF ExtendedProgrammeCounter_TB IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT ExtendedProgrammeCounter
    PORT(
        SR_SB : IN std_logic_vector(9 downto 0);
        ExtendedProgrammeCounter : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;


    --Inputs
    signal SR_SB : std_logic_vector(9 downto 0) := (others => '0');

    --Outputs
    signal ExtendedProgrammeCounter_sig : std_logic_vector(31 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: ExtendedProgrammeCounter PORT MAP (
            SR_SB => SR_SB,
            ExtendedProgrammeCounter => ExtendedProgrammeCounter_sig
        );

    -- Stimulus process
```

```
    stim_proc: process
    begin
        wait for 200ns;
        SR_SB <= "1101011110";

        wait for 200ns;
        SR_SB <= "1111011110";

        wait for 200ns;
        SR_SB <= "0101011010";

        wait for 200ns;
        SR_SB <= "1111111111";

--        wait;
    end process;
END;
```

Control Address Register:

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


entity CAR_TB is
end CAR_TB;

architecture Behavioral of CAR_TB is

    COMPONENT CAR
    Port( CAR_in : in STD_LOGIC_VECTOR(16 downto 0);
        MUX_S, reset: in STD_LOGIC;
        CAR_out : out STD_LOGIC_VECTOR(16 downto 0)
        );
    END COMPONENT;

    --Inputs
    signal CAR_in : std_logic_vector(16 downto 0) := (others => '0');
    signal MUX_S : std_logic := '0';
```

```
    signal reset : std_logic := '0';
    signal clk : std_logic := '0'; -- with clock

    --Outputs
    signal CAR_out : std_logic_vector(16 downto 0);


BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: CAR PORT MAP (
        CAR_in => CAR_in,
        MUX_S => MUX_S,
        reset => reset,
        CAR_out => CAR_out
        );

    -- Stimulus process
    stim_proc: process
    begin

        reset <= '1';
        wait for 150ns;

        reset <= '0';
        wait for 150ns;

        MUX_S <= '1';
        reset <= '0';
        CAR_in <= b"0_0000_0000_0000_1111";
        wait for 150ns;

        MUX_S <= '0';
        CAR_in <= b"0_0000_0000_0000_0000";
        wait for 150ns;


        wait;
    end process;

END;
```

Control Memory:

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

entity ControlMemory_TB is
end ControlMemory_TB;

architecture Behavioral of ControlMemory_TB is

    component ControlMemory
    Port (  FL : out std_logic; -- 0
            RZ : out std_logic; -- 1
            RN : out std_logic; -- 2
            RC : out std_logic; -- 3
            RV : out std_logic; -- 4
            MW : out std_logic; -- 5
            MM : out std_logic; -- 6
            RW : out std_logic; -- 7
            MD : out std_logic; -- 8
            FS : out std_logic_vector(4 downto 0); -- 9 to 13
            MB : out std_logic; -- 14
            TB : out std_logic; -- 15
            TA : out std_logic; -- 16
            TD : out std_logic; -- 17
            PL : out std_logic; -- 18
            PI : out std_logic; -- 19
            IL : out std_logic; -- 20
            MC : out std_logic; -- 21
            MS : out std_logic_vector(2 downto 0); -- 22 to 24
            NA : out std_logic_vector(16 downto 0); -- 25 to 41
            IN_CAR : in std_logic_vector(16 downto 0));
    end component;

    --Inputs
    signal in_car : std_logic_vector(16 downto 0) := (others => '0');

    --Outputs
    signal FL :   std_logic; -- 0
    signal RZ :   std_logic; -- 1
    signal RN :   std_logic; -- 2
```

```
    signal RC :   std_logic; -- 3
    signal RV :   std_logic; -- 4
    signal MW :   std_logic; -- 5
    signal MM :   std_logic; -- 6
    signal RW :   std_logic; -- 7
    signal MD :   std_logic; -- 8
    signal FS :   std_logic_vector(4 downto 0); -- 9 to 13
    signal MB :   std_logic; -- 14
    signal TB :   std_logic; -- 15
    signal TA :   std_logic; -- 16
    signal TD :   std_logic; -- 17
    signal PL :   std_logic; -- 18
    signal PI :   std_logic; -- 19
    signal IL :   std_logic; -- 20
    signal MC :   std_logic; -- 21
    signal MS :   std_logic_vector(2 downto 0); -- 22 to 24
    signal NA :   std_logic_vector(16 downto 0); -- 25 to 41

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: ControlMemory PORT MAP (
            FL => FL,
            RZ => RZ,
            RN => RN,
            RC => RC,
            RV => RV,
            MW => MW,
            MM => MM,
            RW => RW,
            MD => MD,
            FS => FS,
            MB => MB,
            TB => TB,
            TA => TA,
            TD => TD,
            PL => PL,
            PI => PI,
            IL => IL,
            MC => MC,
            MS => MS,
            NA => NA,
```

```
            IN_CAR => IN_CAR
            );

    stim_proc: process
    begin
        wait for 60ns;
        in_car <= "0" & x"0000"; -- 00

        wait for 60ns;
        in_car <= "0" & x"0001"; -- 01

        wait for 60ns;
        in_car <= "0" & x"0002"; -- 02

        wait for 60ns;
        in_car <= "0" & x"0003"; -- 03

        wait for 60ns;
        in_car <= "0" & x"0004"; -- 04

        wait for 60ns;
        in_car <= "0" & x"0005"; -- 05

        wait for 60ns;
        in_car <= "0" & x"0006"; -- 06


        wait for 60ns;
        in_car <= "0" & x"0007"; -- 07


        wait for 60ns;
        in_car <= "0" & x"0008"; -- 08

        wait for 60ns;
        in_car <= "0" & x"0009"; -- 09

        wait for 60ns;
        in_car <= "0" & x"00C0"; -- c0

        wait for 60ns;
        in_car <= "0" & x"00C1"; -- c1
```

```
        wait;
    end process;
END;
```

Memory Module:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Memory_TB IS
END Memory_TB;

ARCHITECTURE behavior OF Memory_TB IS

    component memory is -- use unsigned for memory address
    Port ( address : in std_logic_vector(31 downto 0);
           write_data : in std_logic_vector(31 downto 0);
           MemWrite : in std_logic;
           clk : in std_logic;
           read_data : out std_logic_vector(31 downto 0)
           );
    end component;

    --Inputs
    signal address :  std_logic_vector(31 downto 0) := (others => '0');
    signal write_data :  std_logic_vector(31 downto 0) := (others => '0');
    signal MemWrite:  std_logic := '0';
    signal clk:  std_logic := '0';

    --Outputs
    signal read_data :  std_logic_vector(31 downto 0);

begin
    -- Instantiate the Unit Under Test (UUT)
    uut: memory PORT MAP (
           address => address,
           write_data => write_data,
           MemWrite => MemWrite,
           clk => clk,
           read_data => read_data
           );
```

```
    -- Stimulus process
    clk <= not clk after 10ns;
    stim_proc: process
    begin

    wait for 50ns;

    address <= x"00000001";
    wait for 50ns;

    address <= x"00000002";
    wait for 50ns;

    address <= x"00000003";
    wait for 50ns;

    address <= x"00000000";
    write_data <= x"00000000";
    MemWrite <= '1';
    wait for 50ns;

    address <= x"00000001";
    write_data <= x"00000001";
    wait for 50ns;

    address <= x"00000002";
    write_data <= x"00000002";
    wait for 50ns;

    address <= x"00000003";
    write_data <= x"00000003";
    wait for 50ns;

    address <= x"00000000";
    MemWrite <= '0';
    wait for 50ns;

    address <= x"00000001";
    wait for 50ns;

    address <= x"00000002";
```

```
    wait for 50ns;

    address <= x"00000003";
    wait for 50ns;

    wait;
  end process;

  END;
```

Micro Programme Controller:

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


entity MicroProgrammeController_TB is
end MicroProgrammeController_TB;

architecture Behavioral of MicroProgrammeController_TB is

component MicroprogrammeController is
    Port(   IR : in STD_LOGIC_VECTOR(31 downto 0);
            status_bits : in STD_LOGIC_VECTOR(3 downto 0);
            reset_mpc : in STD_LOGIC;
            clk_mpc : in STD_LOGIC;
            control_word_mpc : out STD_LOGIC_VECTOR(23 downto 0);
            PC_out : out STD_LOGIC_VECTOR(31 downto 0);
            TD_mpc, TA_mpc, TB_mpc, MW_mpc : out STD_LOGIC
        );
end component;

signal IR : STD_LOGIC_VECTOR(31 downto 0):= (others => '0');
signal status_bits : STD_LOGIC_VECTOR(3 downto 0):=(others=>'0');
signal reset_mpc : STD_LOGIC :='0';
signal clk_mpc : STD_LOGIC :='0';

signal control_word_mpc : STD_LOGIC_VECTOR(23 downto 0);
signal PC_out : STD_LOGIC_VECTOR(31 downto 0);
signal TD_mpc, TA_mpc, TB_mpc, MW_mpc : STD_LOGIC;
```

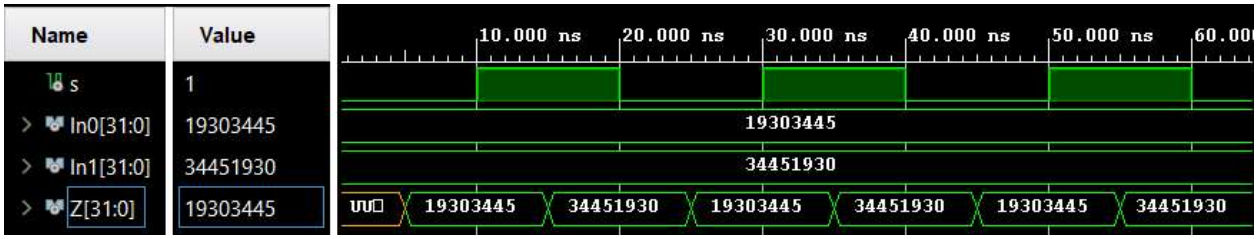
```
begin
   -- Instantiate the Unit Under Test (UUT)
    uut: MicroprogrammeController PORT MAP (
        IR => IR,
        status_bits => status_bits,
        reset_mpc => reset_mpc,
        clk_mpc => clk_mpc,
        control_word_mpc => control_word_mpc,
        PC_out => PC_out,
        TD_mpc => TD_mpc,
        TA_mpc => TA_mpc,
        TB_mpc => TB_mpc,
        MW_mpc => MW_mpc
    );


    clk_mpc <= not clk_mpc after 25ns;
    -- Stimulus process
    stim_proc: process
    begin
        reset_mpc <= '1';
        wait for 100ns;

        reset_mpc <= '0';
        wait for 100ns;

        IR <= x"ffffffff";
        status_bits <= x"f";

--        wait;
    end process;

end Behavioral;
```

Processor:

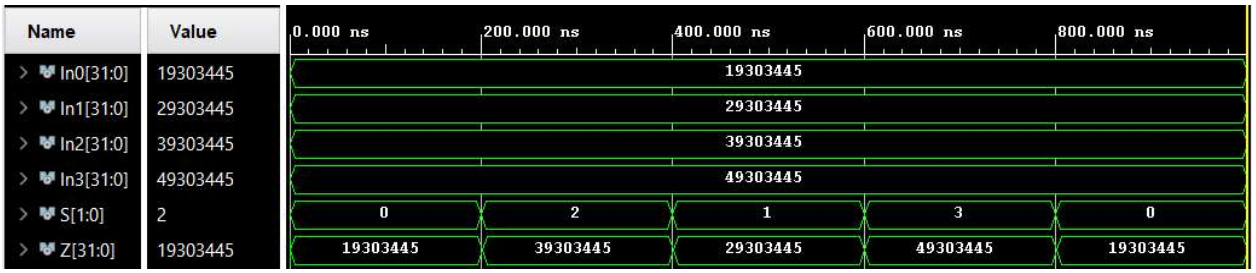
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Processor_TB is
end Processor_TB;

architecture Simulation of Processor_TB is

    component Processor is
        Port( Clk, reset : in STD_LOGIC );
    end component;

    --Inputs
    signal Clk : std_logic := '0';
    signal reset : std_logic := '0';

begin

    -- Instantiate the Unit Under Test (UUT)
    uut: Processor PORT MAP (
        Clk => Clk,
        reset => reset
    );


    clk <= not clk after 50ns;
    -- Stimulus process
    stim_proc: process
    begin

        reset <= '1';
        wait for 100ns;

        reset <= '0';
        wait for 600ns;

--        wait;
    end process;
END;
```

ScreenShots

MUX 2 32 Bit:

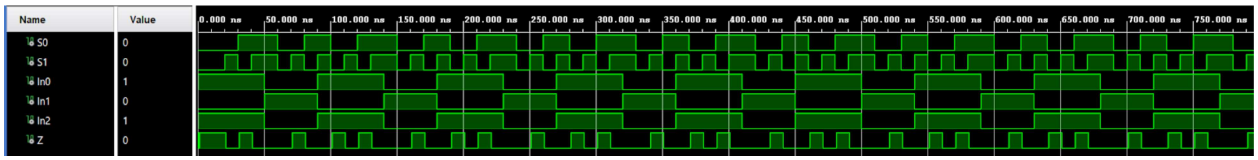


MUX 4 32 Bit:

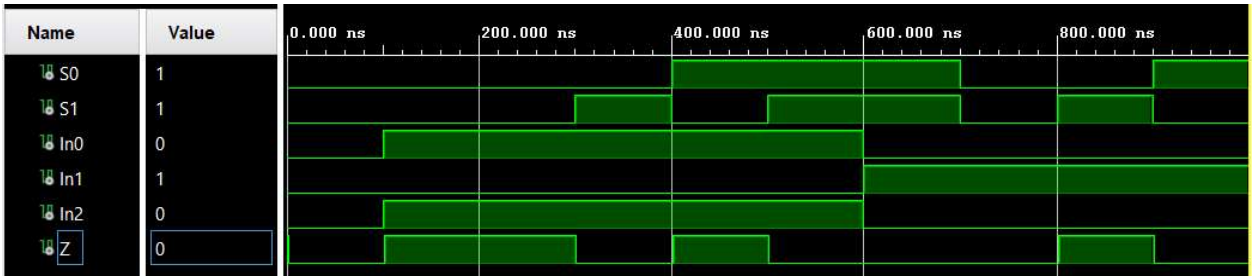


MUX 3 1 Bit

Zoomed out with less wait time

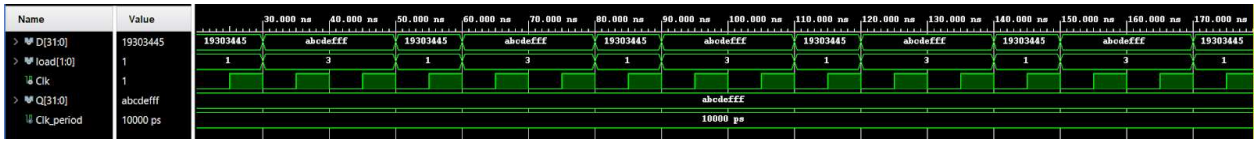


Zoomed in with more wait time

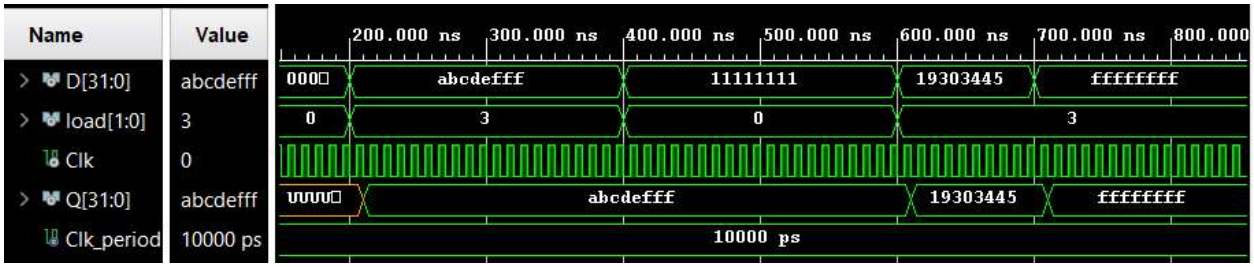


Register 32 Bit:

Zoomed out with less wait time

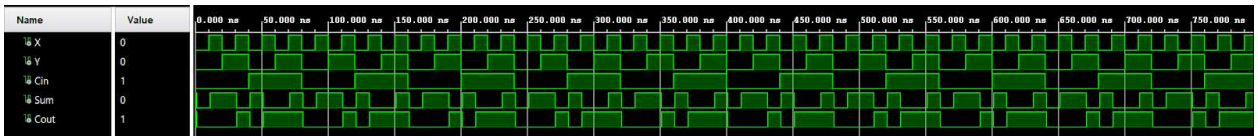


Zoomed in with more wait time

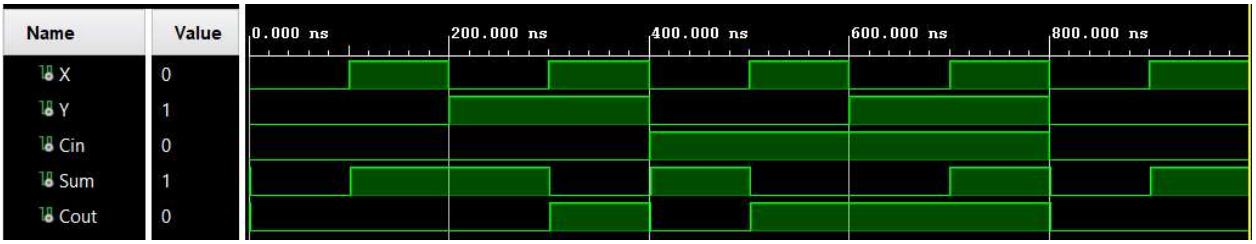


Full Adder:

Zoomed out with less wait time (10 ns)



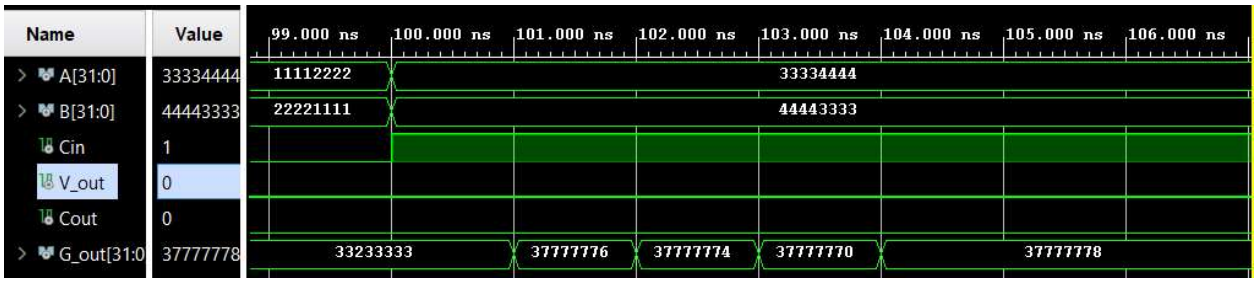
Zoomed in with more wait time (100 ns)

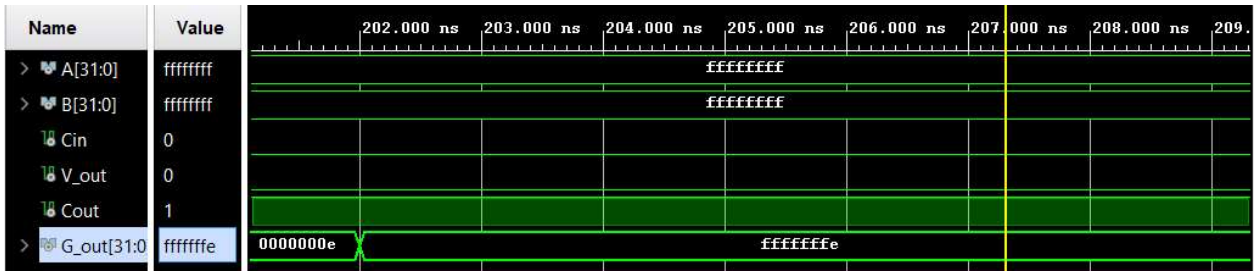


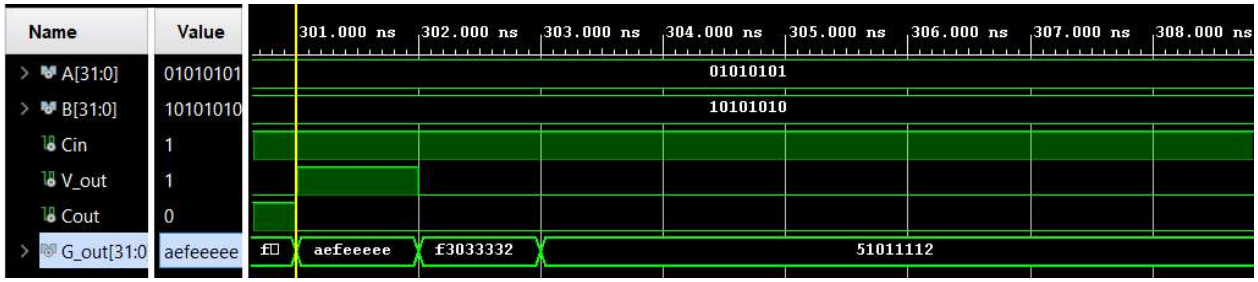
Ripple Adder:

Zoomed out with less wait time (10 ns)

Zoomed in with 100 ns wait time

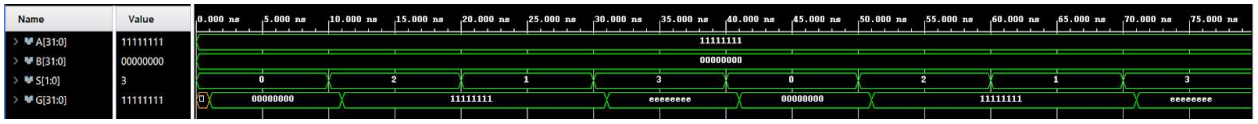




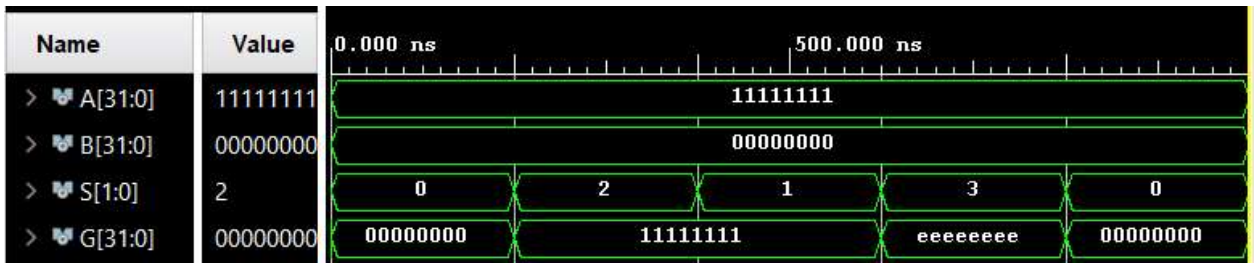


Logic Unit:

Zoomed out with 10 ns wait time



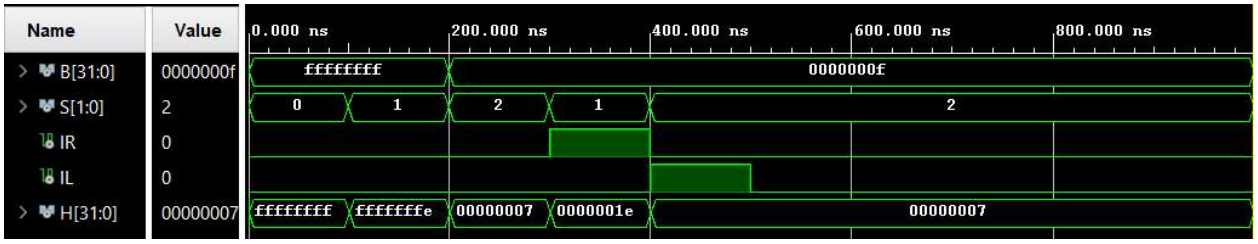
Zoomed in with 100 ns wait time



Shifter :

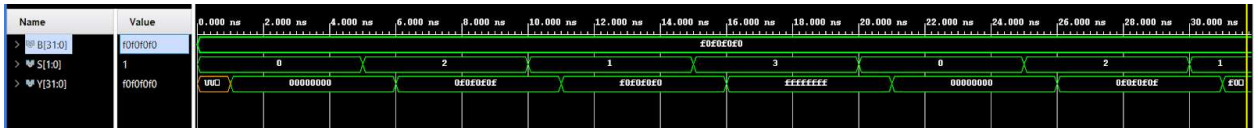
Zoomed out with 10 ns wait time

Zoomed in with 100 ns wait time

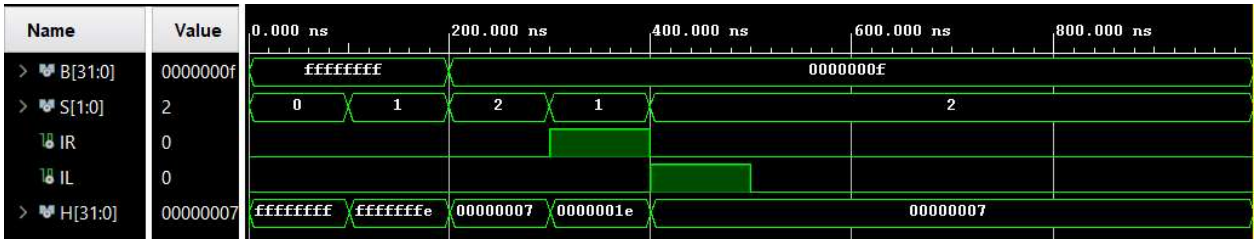


B input logic:

Zoomed out with 10 ns wait time

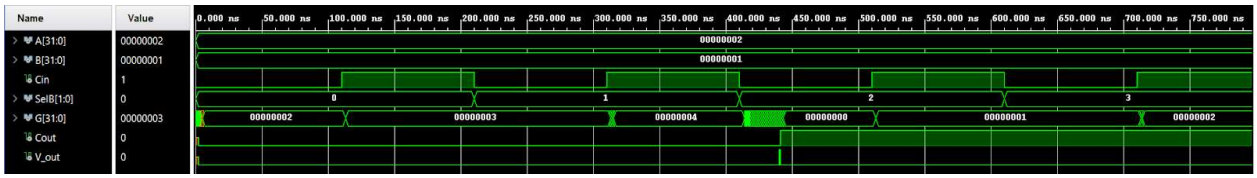


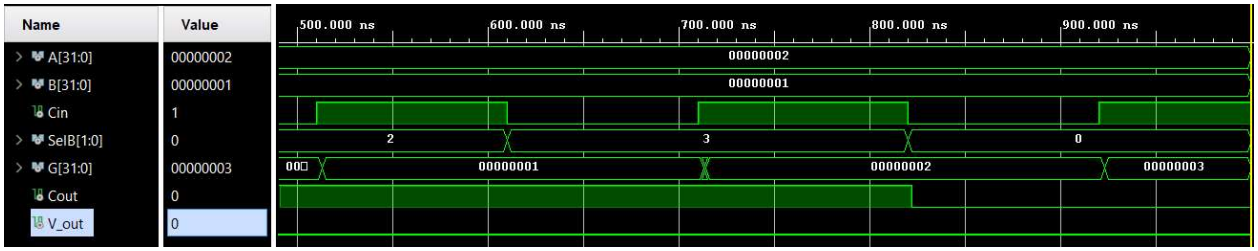
Zoomed in with 100 ns wait time



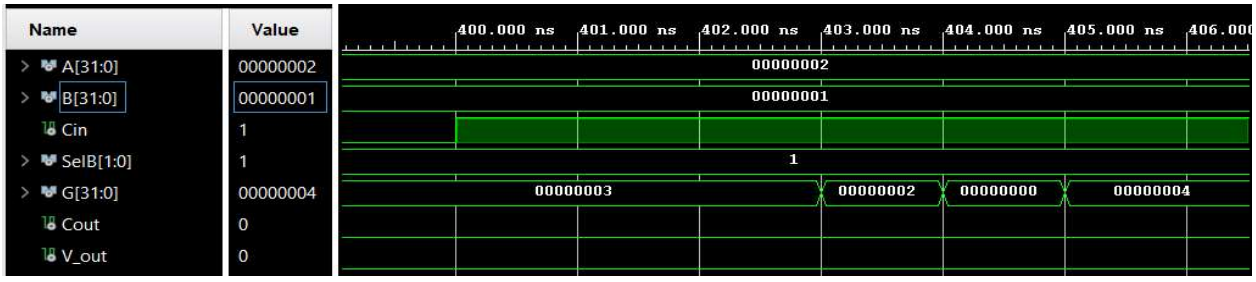
Arithmetic Unit:

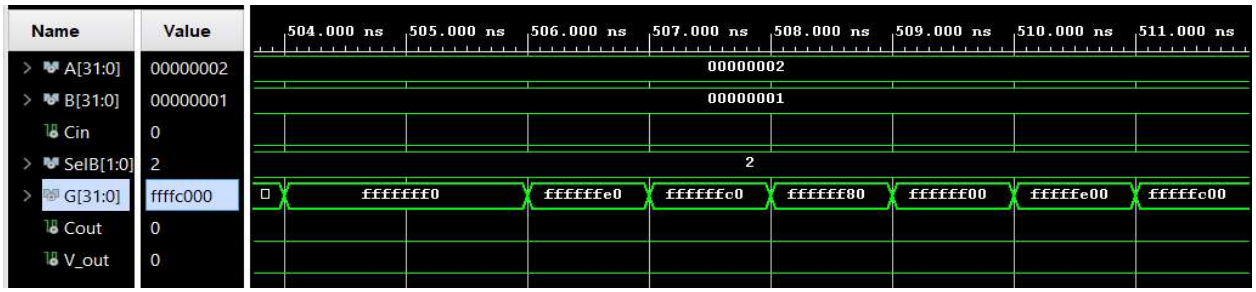
Zoomed out with 100 ns wait time

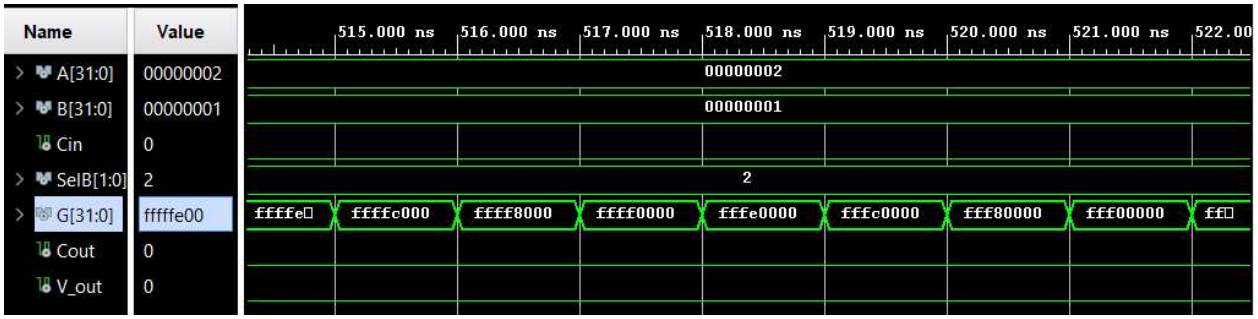


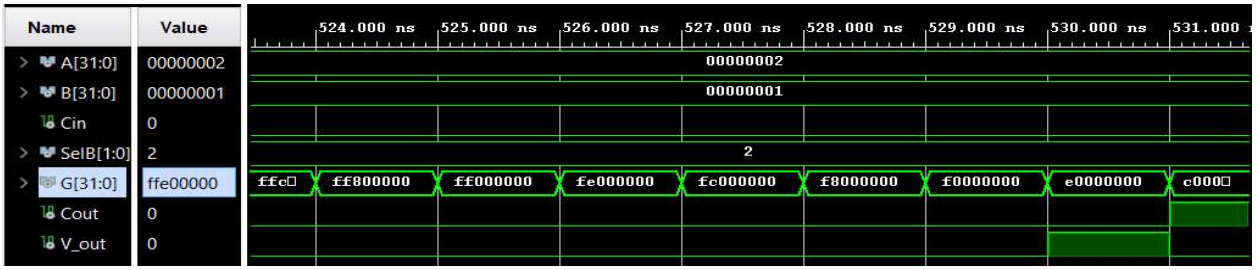


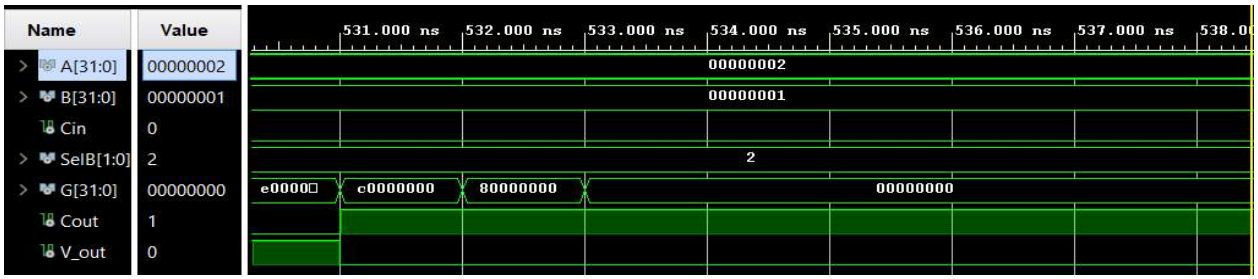
Zoomed in with the same wait time





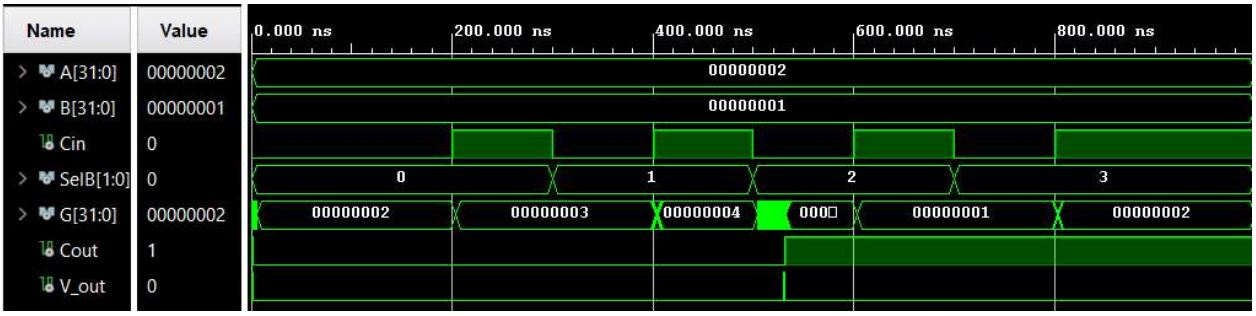




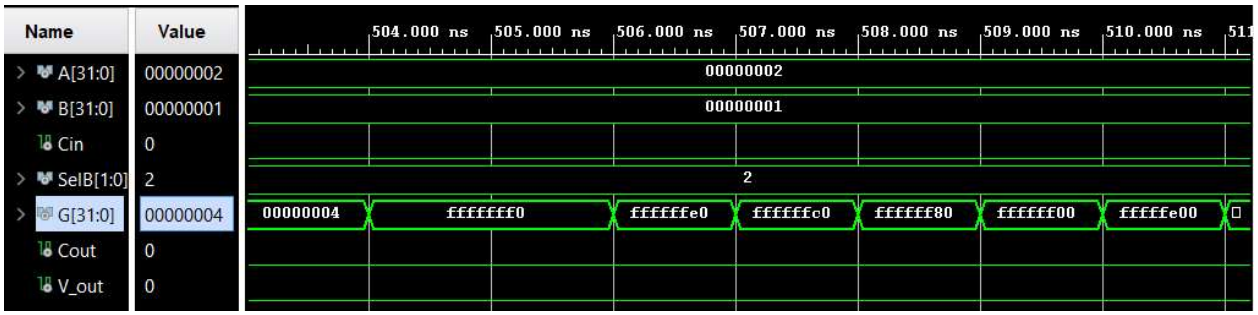


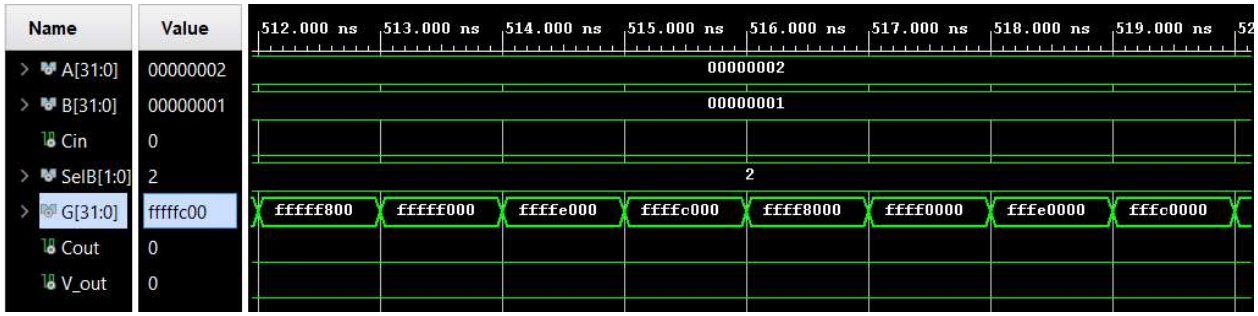
ALU:

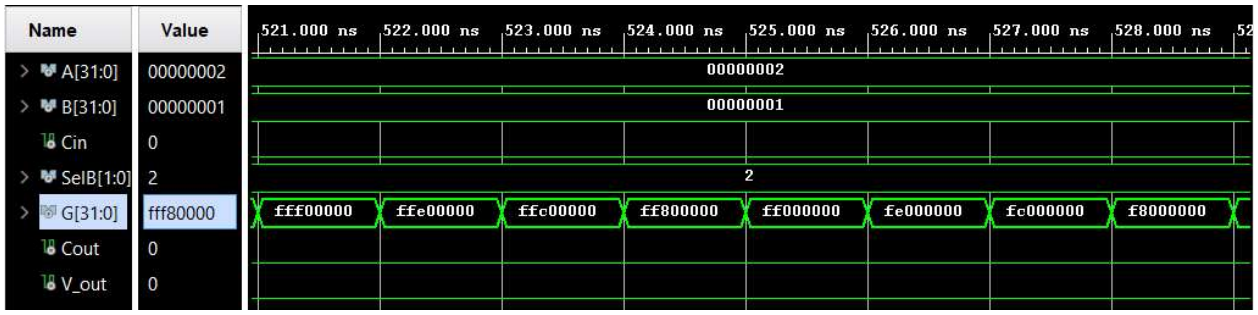
Zoomed out with 75 ns wait time

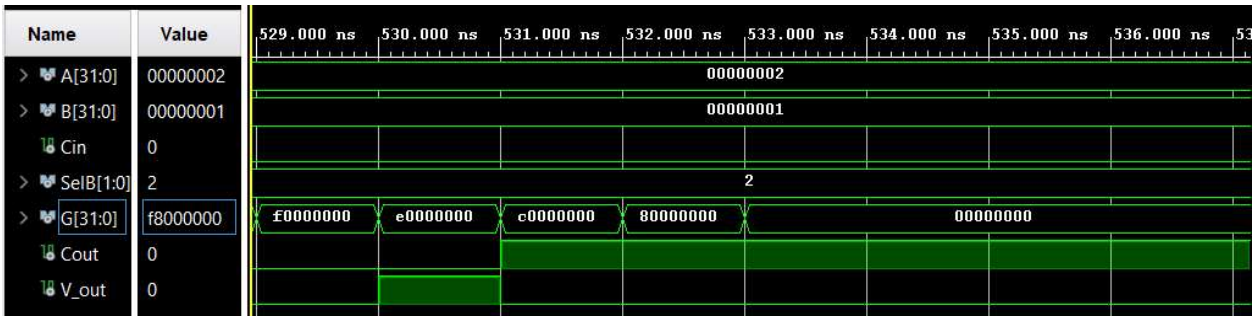


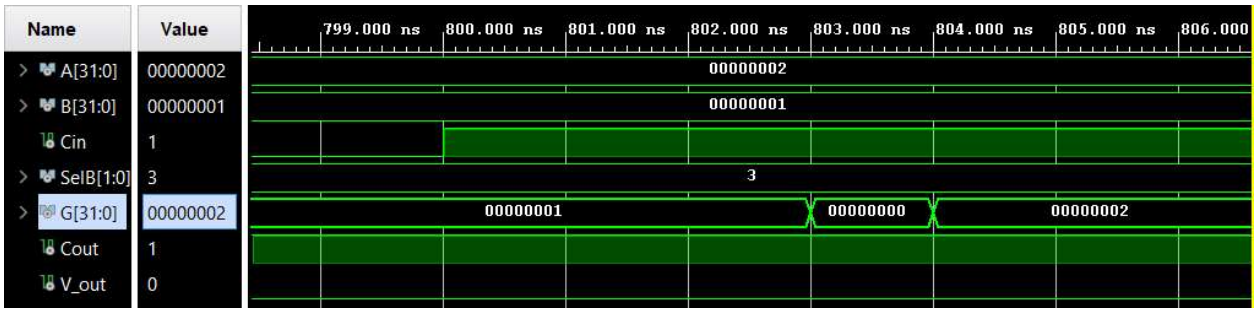
Zoomed in with same wait time.





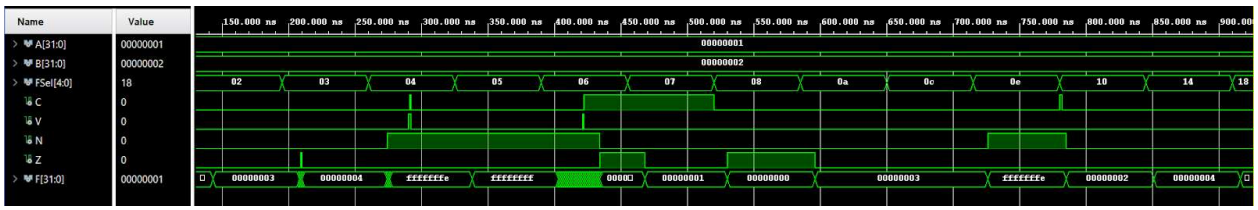


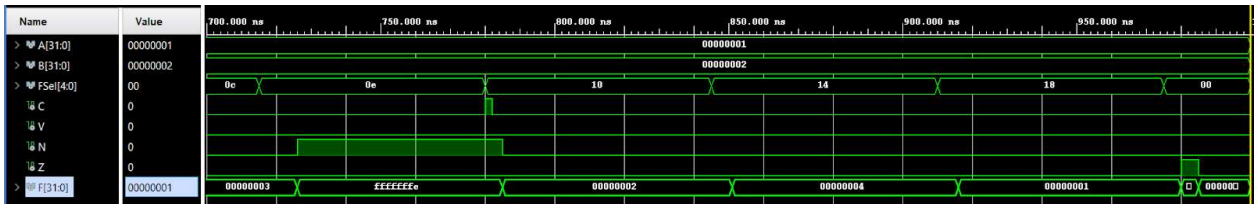




Function Unit:

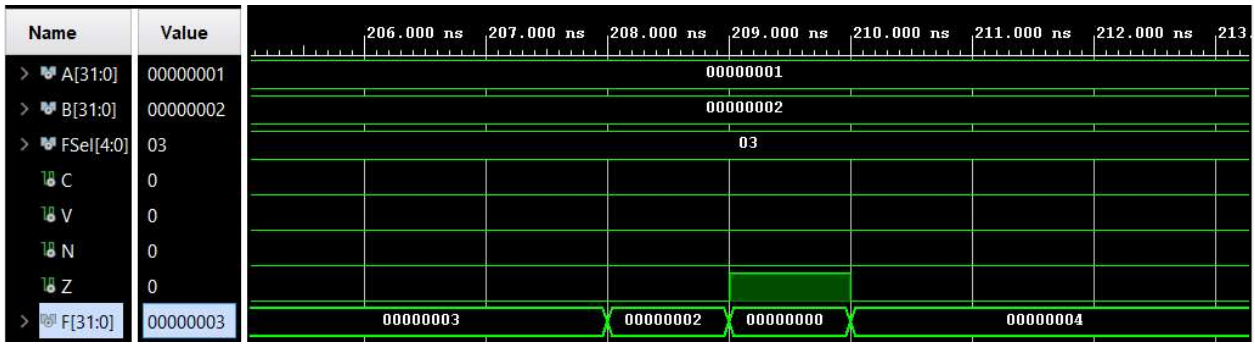
Zoomed out with 65 ns wait time



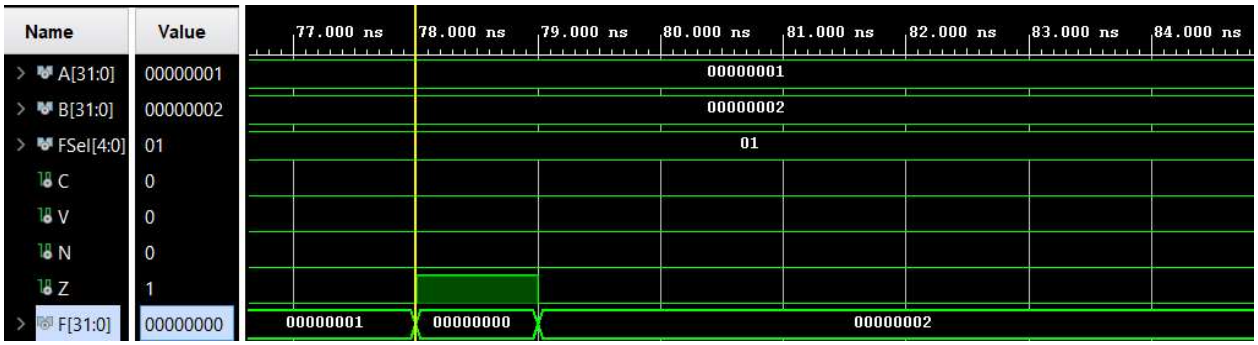


Zoomed in with the same wait time

Addition with carry

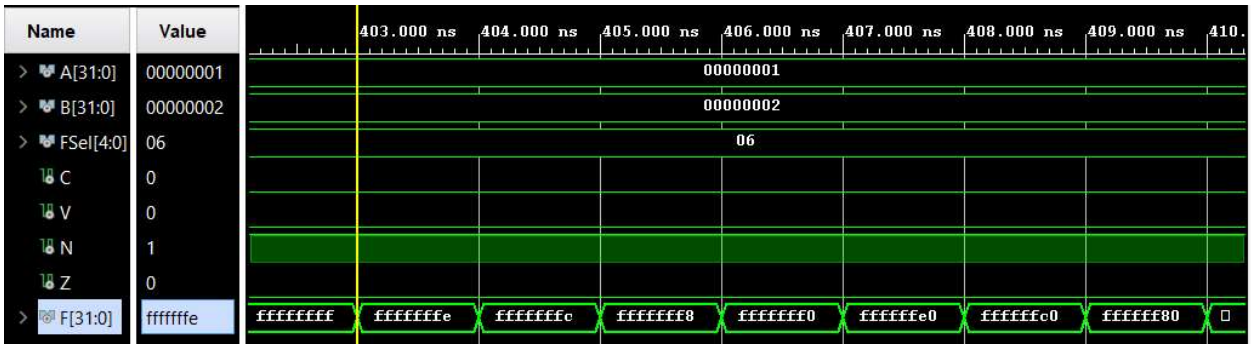


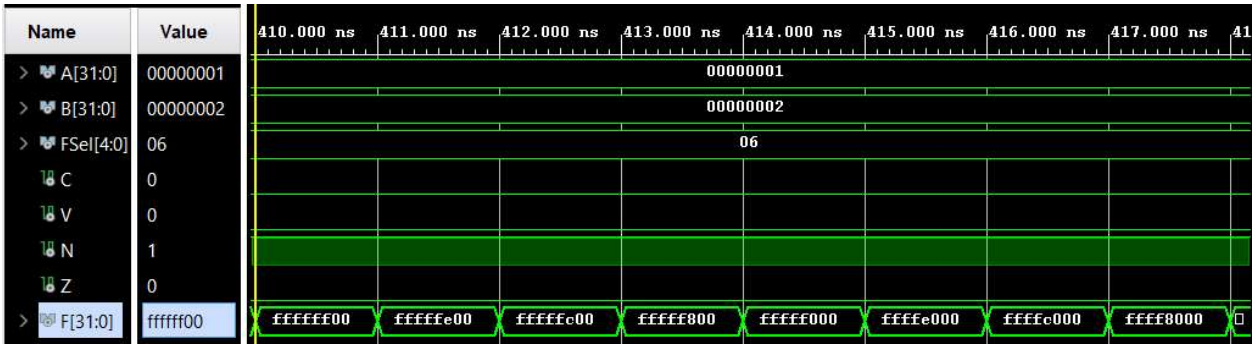
F = A + 1

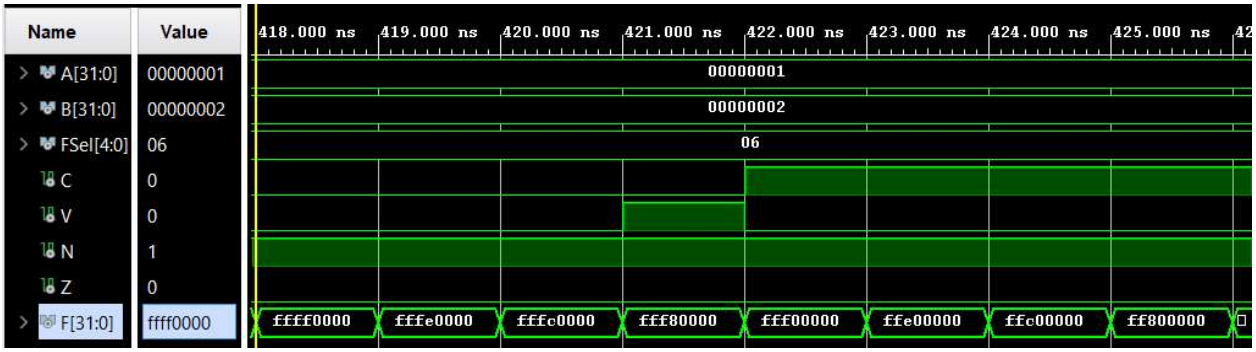


Addition WITH NOT B

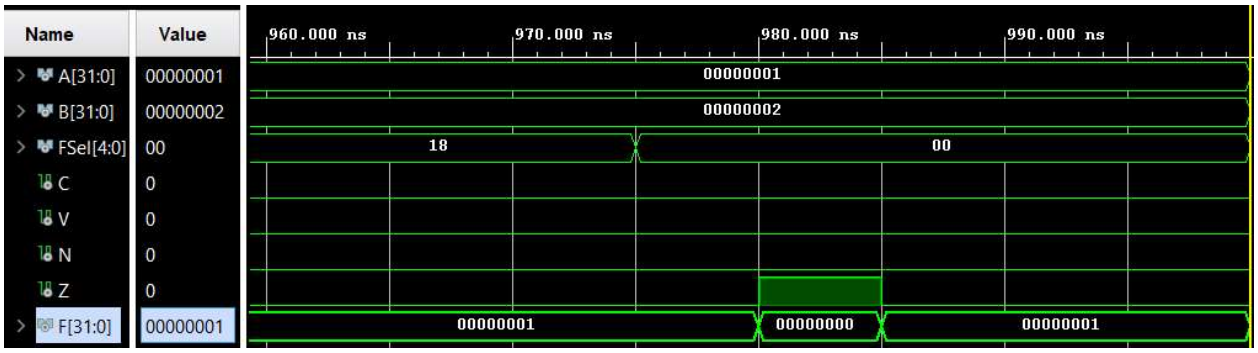
Decrement


Name
Value
A[31:0]
00000001
B[31:0]
00000002
FSel[4:0]
06
C
0
V
0
N
1
Z
0
F[31:0]
fffffffe
403.000 ns
404.000 ns
405.000 ns
406.000 ns
407.000 ns
408.000 ns
409.000 ns
ffffffff
fffffffe
fffffffc
fffffff8
fffffff0
ffffffe0
ffffffc0
ffffff80


Name
Value
A[31:0]
00000001
B[31:0]
00000002
FSel[4:0]
06
C
0
V
0
N
1
Z
0
F[31:0]
ffffff00
410.000 ns
411.000 ns
412.000 ns
413.000 ns
414.000 ns
415.000 ns
416.000 ns
417.000 ns
ffffff00
fffffe00
fffffc00
fffff800
fffff000
ffffe000
ffffc000
ffff8000

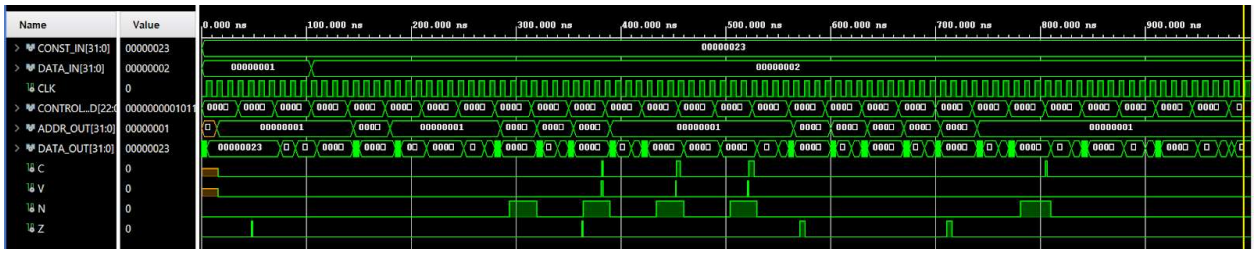

Name
Value
A[31:0]
00000001
B[31:0]
00000002
FSel[4:0]
06
C
0
V
0
N
1
Z
0
F[31:0]
ffff0000
418.000 ns
419.000 ns
420.000 ns
421.000 ns
422.000 ns
423.000 ns
424.000 ns
425.000 ns
ffff0000
fffe0000
fffc0000
fff80000
fff00000
ffe00000
ffc00000
ff800000


Name
Value
A[31:0]
00000001
B[31:0]
00000002
FSel[4:0]
06
C
1
V
0
N
1
Z
0
F[31:0]
ff000000
426.000 ns
427.000 ns
428.000 ns
429.000 ns
430.000 ns
431.000 ns
432.000 ns
433.000 ns
ff000000
fe000000
fc000000
f8000000
f0000000
e0000000
c0000000
80000000

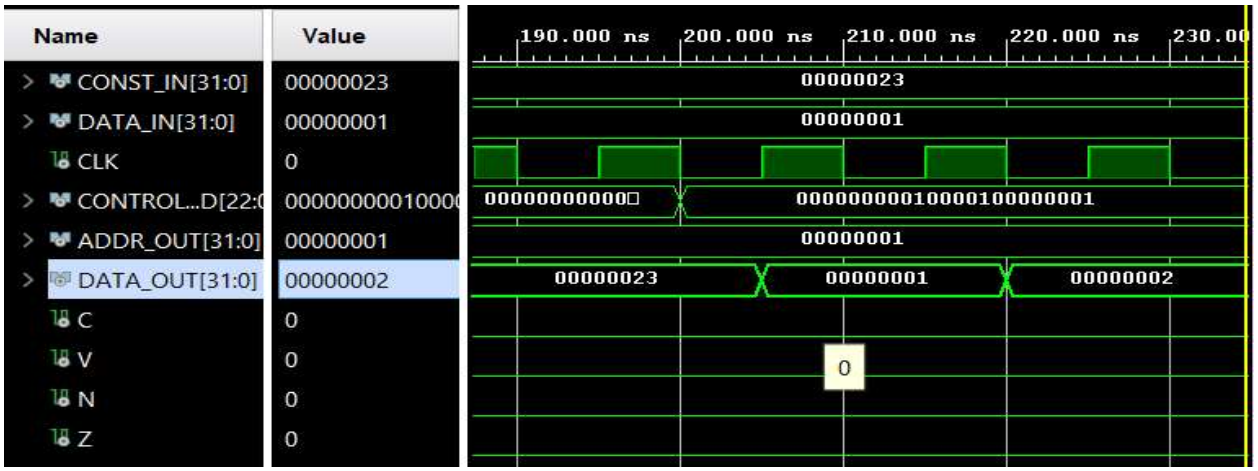


Old Datapath:

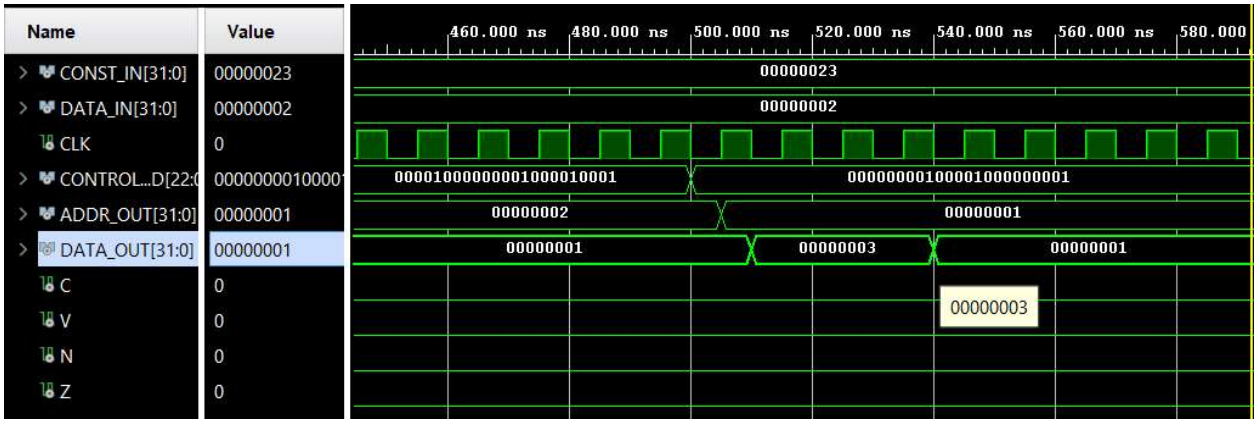
Zoomed out



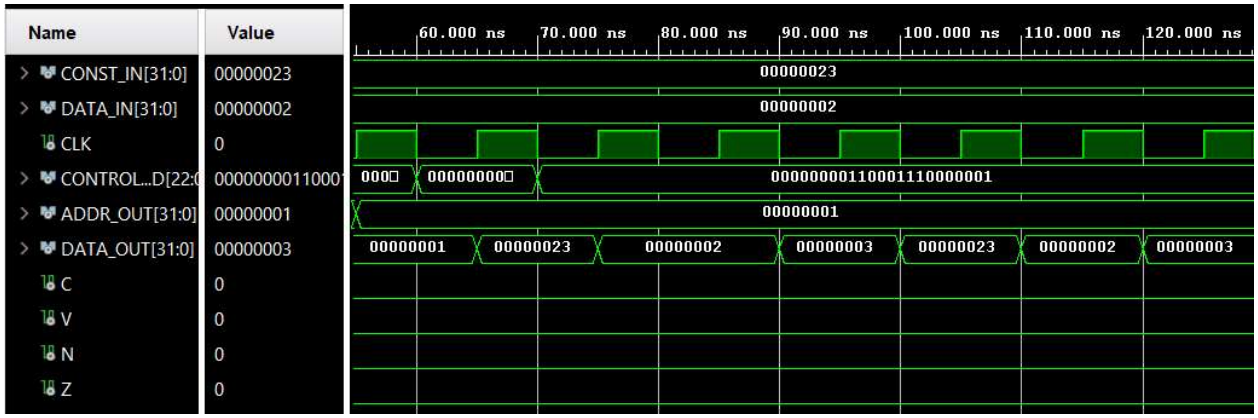
F = A + 1



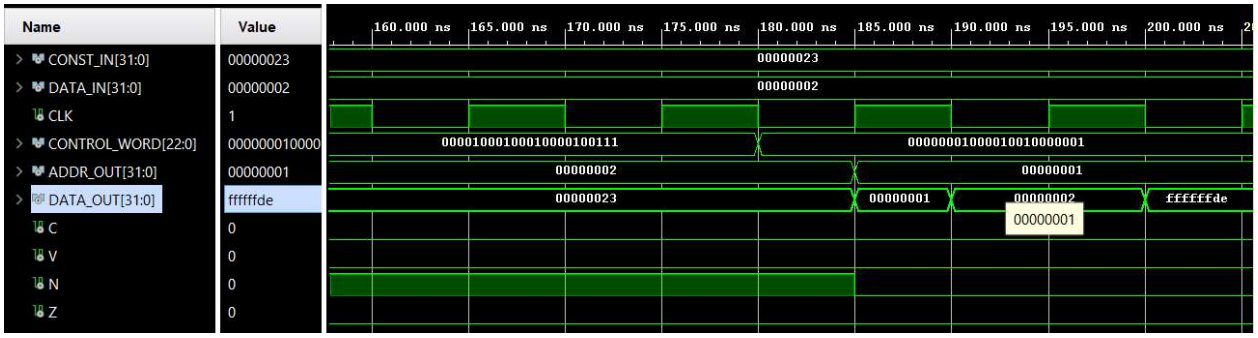
F = A + B


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
0
CONTROL...D[22:0]
0000000010000
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000001
C
0
V
0
N
0
Z
0
460.000 ns
480.000 ns
500.000 ns
520.000 ns
540.000 ns
560.000 ns
580.000
00000023
00000002
00000001000010001000010001
00000000100001000000001
00000002
00000001
00000001
00000003
00000001
00000003

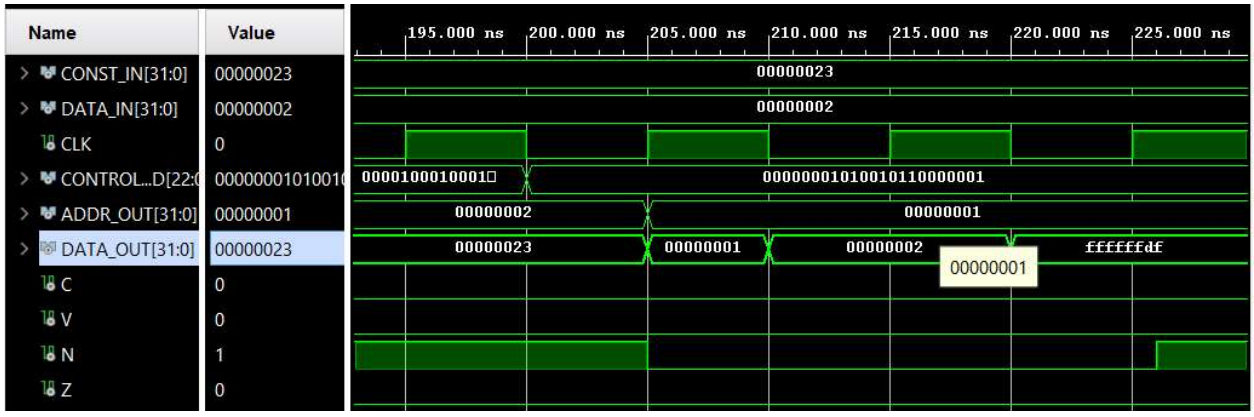
F = A + B + 1


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
0
CONTROL...D[22:0]
0000000011000
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000003
C
0
V
0
N
0
Z
0
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns
120.000 ns
00000023
00000002
000
00000000
00000000110001110000001
00000001
00000001
00000023
00000002
00000003
00000023
00000002
00000003

F = A + (~B)


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
1
CONTROL_WORD[22:0]
000000010000
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
fffffde
C
0
V
0
N
0
Z
0
160.000 ns
165.000 ns
170.000 ns
175.000 ns
180.000 ns
185.000 ns
190.000 ns
195.000 ns
200.000 ns
00000023
00000002
00001000100010000100111
00000001000010010000001
00000002
00000001
00000023
00000001
00000002
fffffffde
00000001

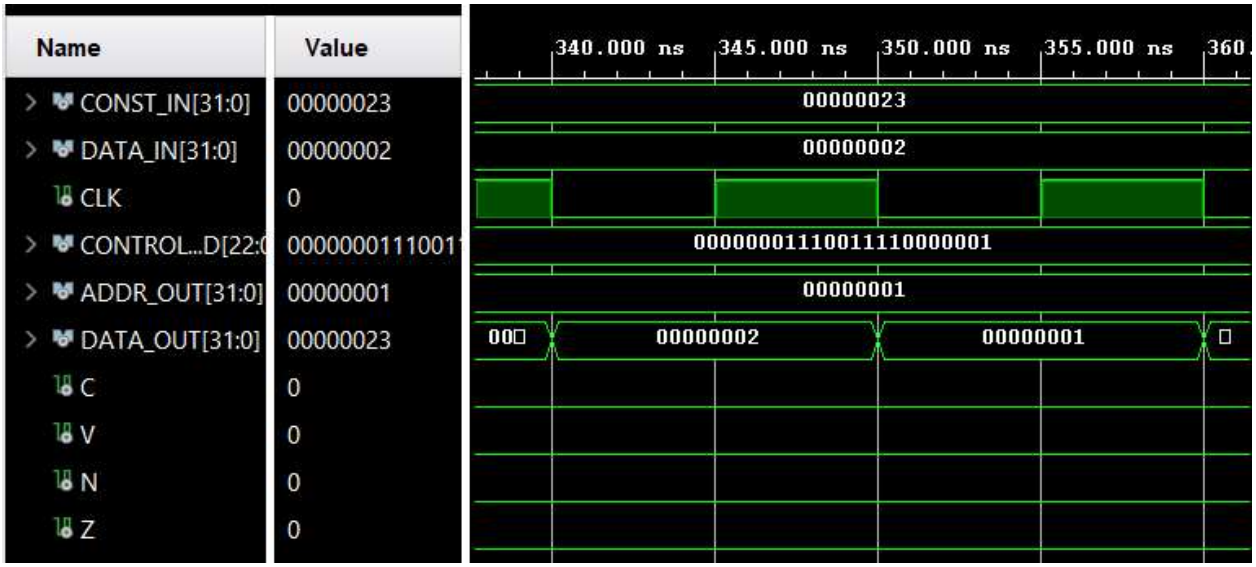
F = A + (~B) - 1


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
0
CONTROL...D[22:0]
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000023
C
0
V
0
N
1
Z
0
195.000 ns
200.000 ns
205.000 ns
210.000 ns
215.000 ns
220.000 ns
225.000 ns
00000023
00000002
00000001
00000002
00000023
00000001
00000002
00000001
ffffffdf

F = A - 1


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
1
CONTROL...D[22:0]
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000023
C
0
V
0
N
0
Z
0
280.000 ns
290.000 ns
300.000 ns
310.000 ns
00000023
00000002
00000001
00000000
00000023
00000002
00000000
00000023

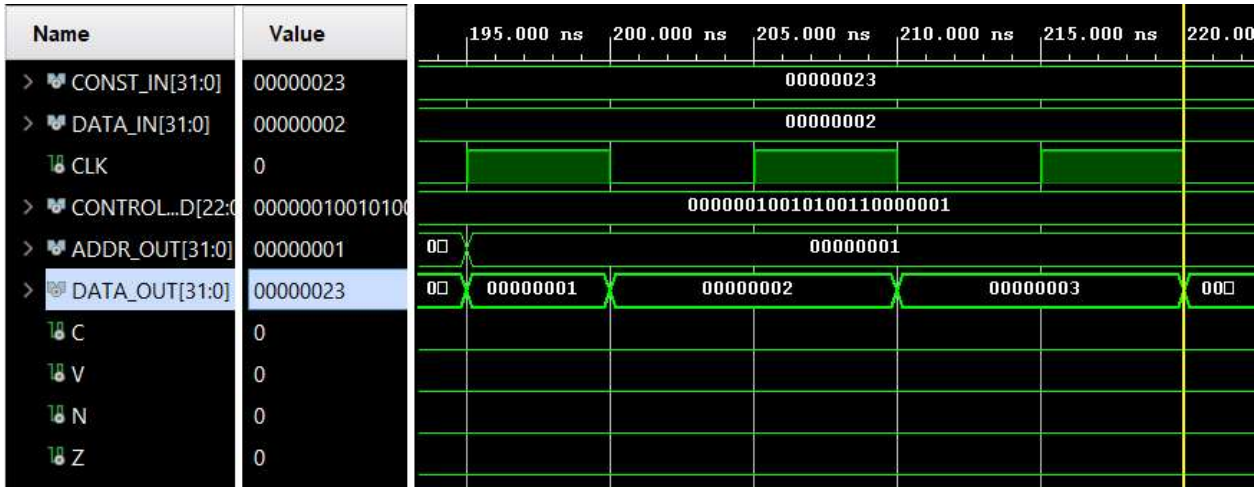
F = A


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
0
CONTROL...D[22:0]
00000001110011
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000023
C
0
V
0
N
0
Z
0
340.000 ns
345.000 ns
350.000 ns
355.000 ns
00000023
00000002
00000001110011110000001
00000001
00000002
00000001

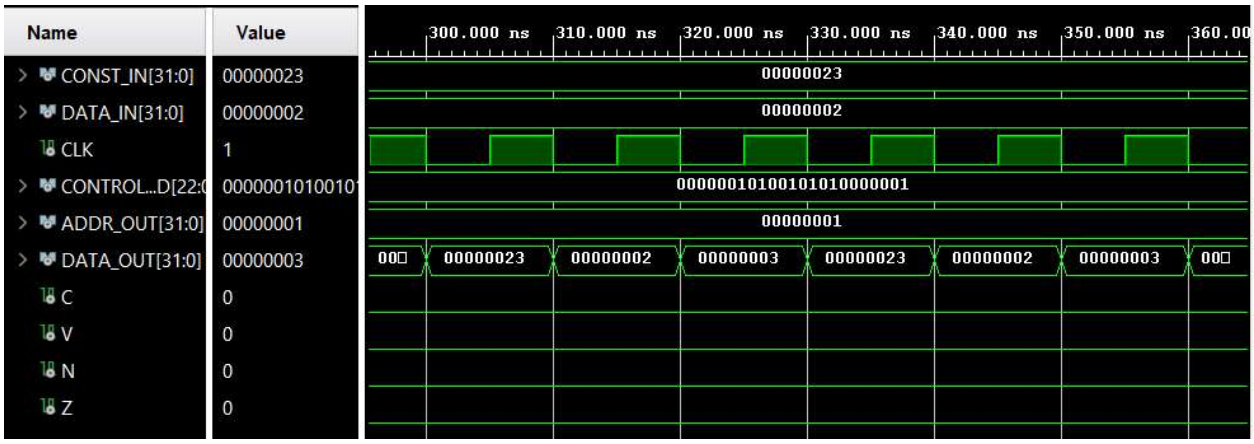
F = A ^ B


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
1
CONTROL...D[22:0]
00000010000100
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000002
C
0
V
0
N
0
Z
0
280.000 ns
290.000 ns
300.000 ns
310.000 ns
00000023
00000002
00000001000010001000001
00000001
00000002
00000000
00000023
00000002
00000000

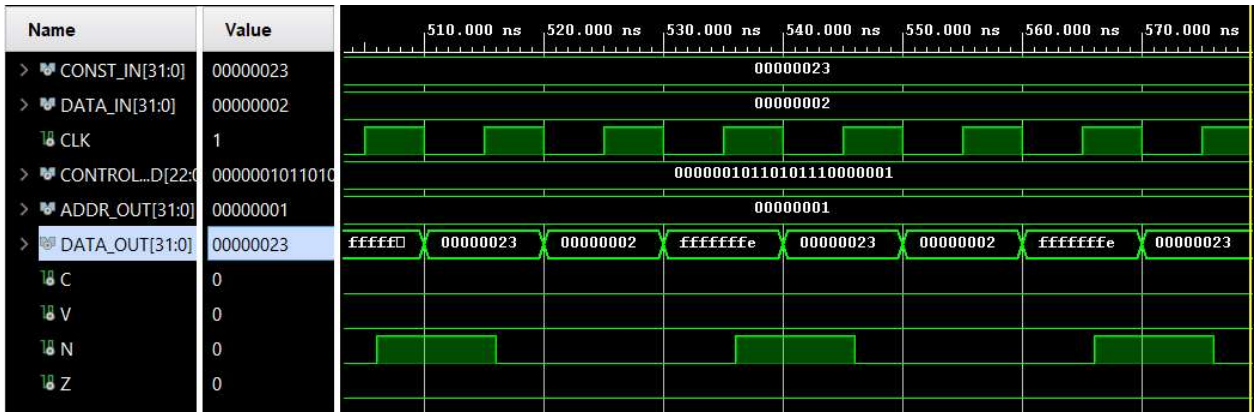
F = A V B


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
0
CONTROL...D[22:0]
00000010010100
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000023
C
0
V
0
N
0
Z
0
195.000 ns
200.000 ns
205.000 ns
210.000 ns
215.000 ns
220.00
00000023
00000002
00000010010100110000001
00000001
00000001
00000002
00000003

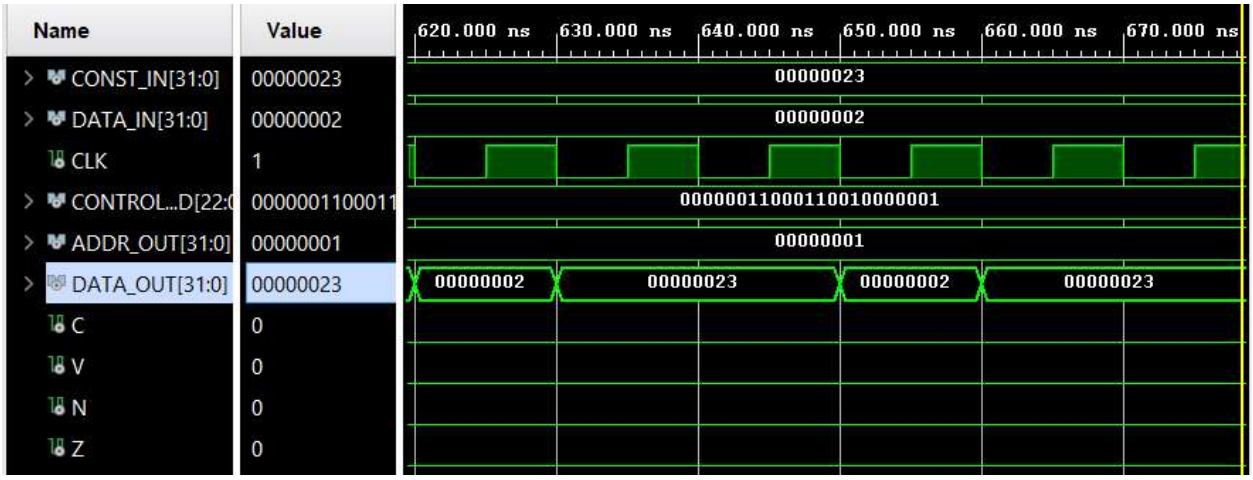
F = A ⊕ B


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
1
CONTROL...D[22:0]
00000010100101
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000003
C
0
V
0
N
0
Z
0
300.000 ns
310.000 ns
320.000 ns
330.000 ns
340.000 ns
350.000 ns
360.00
00000023
00000002
00000010100101010000001
00000001
00000023
00000002
00000003
00000023
00000002
00000003

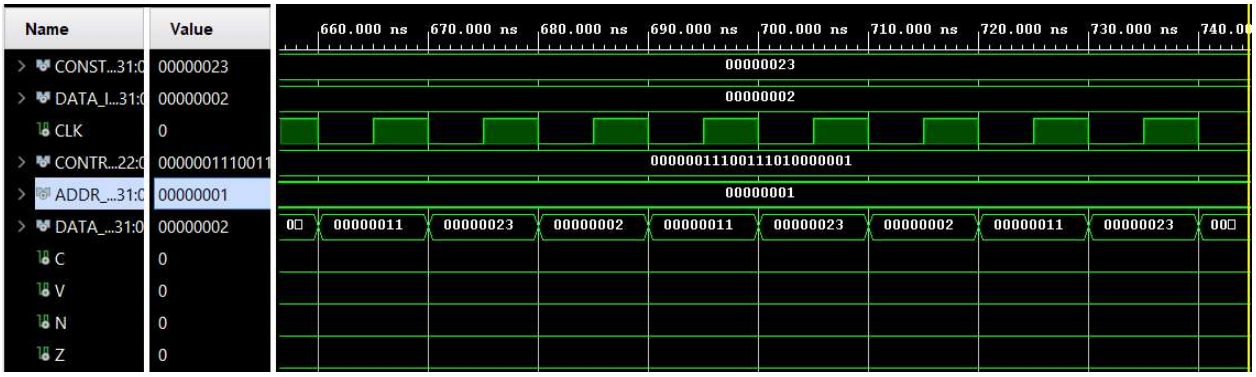
F = (~A)


Name
Value
CONST_IN[31:0]
00000023
DATA_IN[31:0]
00000002
CLK
1
CONTROL...D[22:0]
00000010110101
ADDR_OUT[31:0]
00000001
DATA_OUT[31:0]
00000023
C
0
V
0
N
0
Z
0
510.000 ns
520.000 ns
530.000 ns
540.000 ns
550.000 ns
560.000 ns
570.000 ns
00000023
00000002
00000010110101110000001
00000001
00000023
00000002
fffffffe
00000023
00000002
fffffffe
00000023

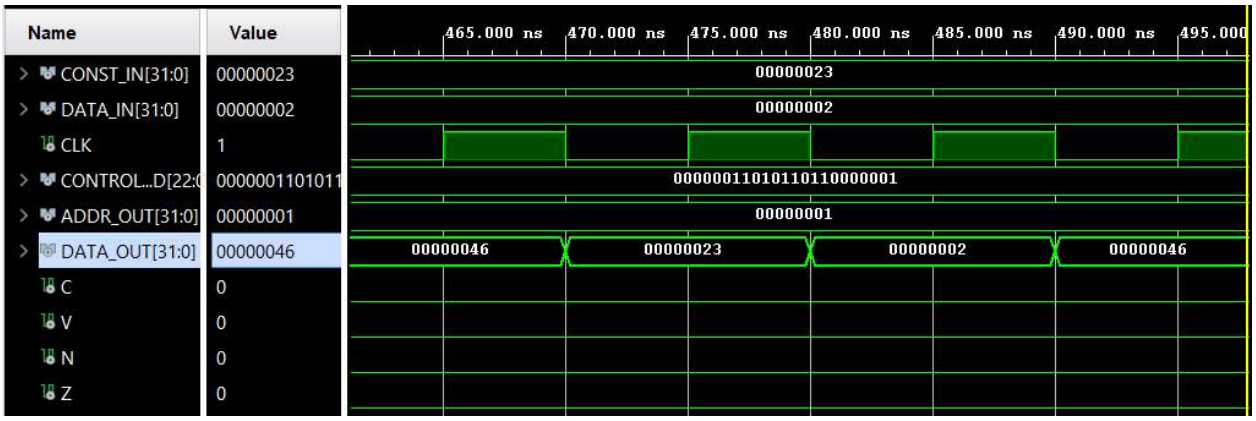
F = B



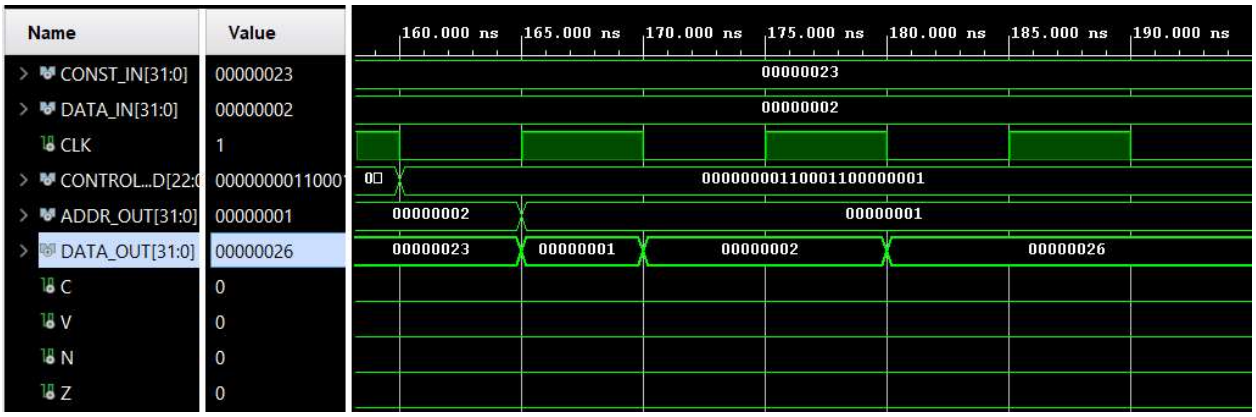
F = sr B



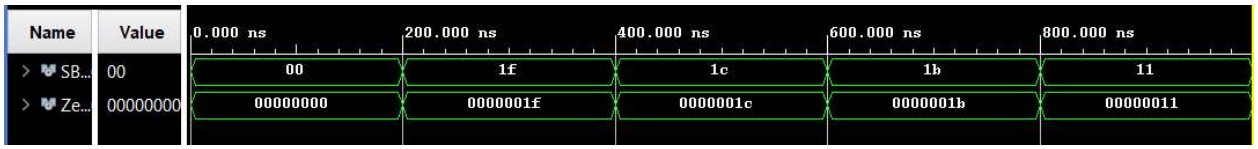
F = sl B



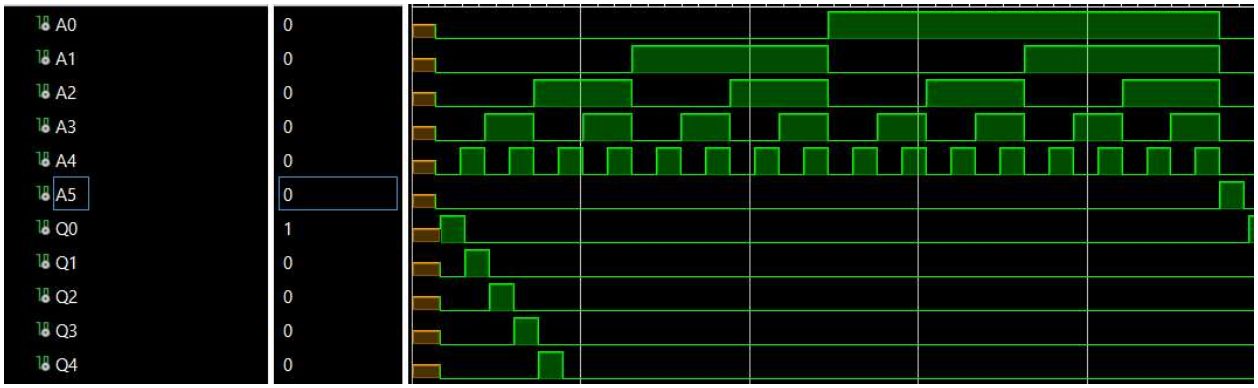
F = A + B + CONST_IN

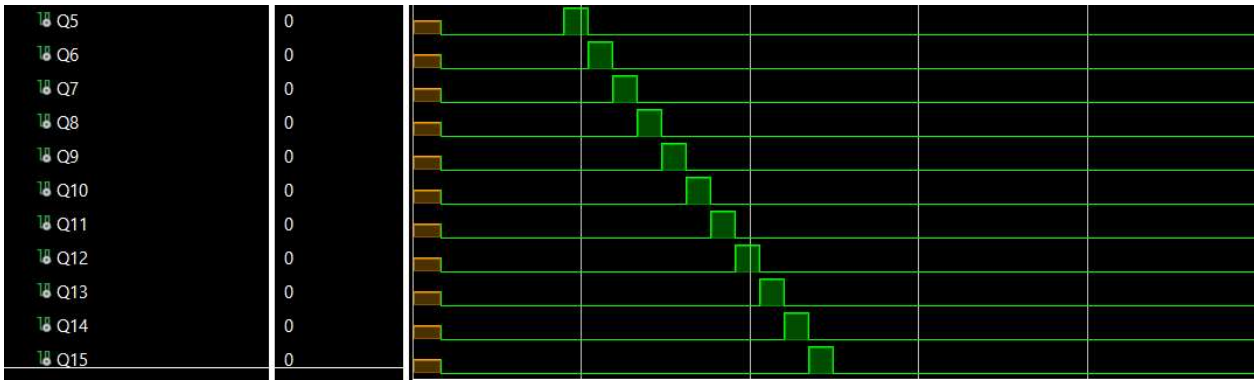


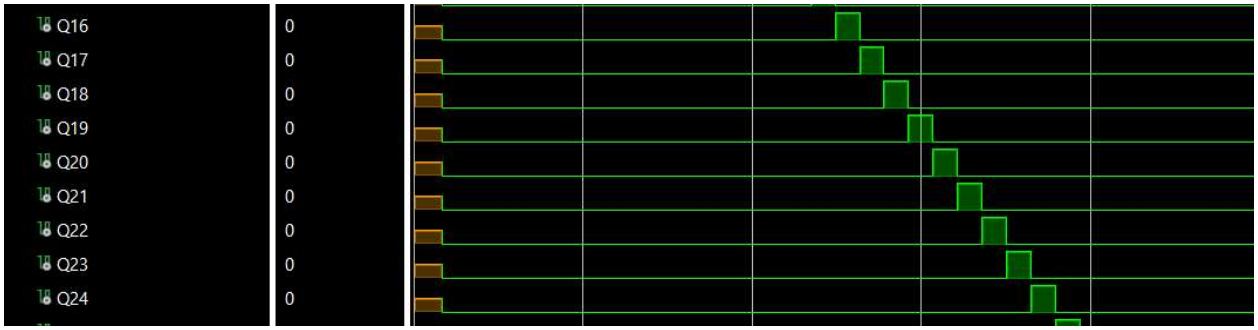
Zero Fill:

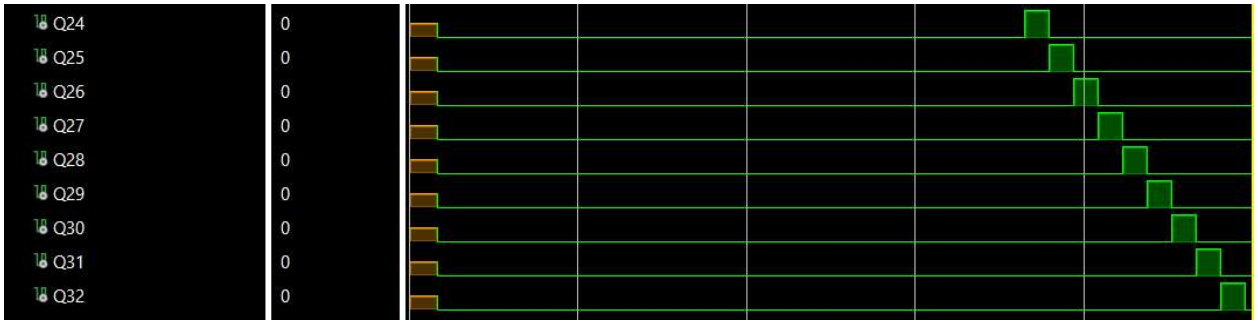


Decoder_6to33:

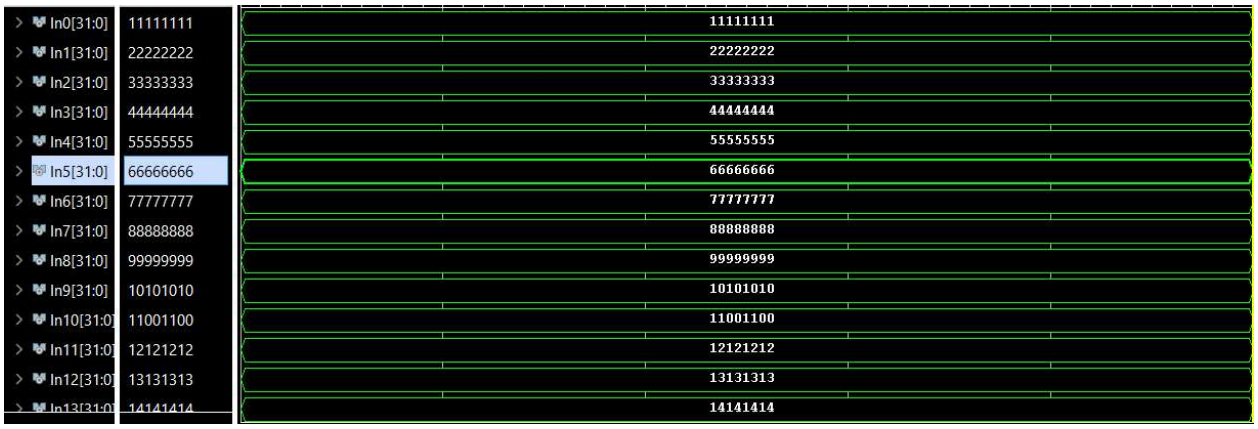


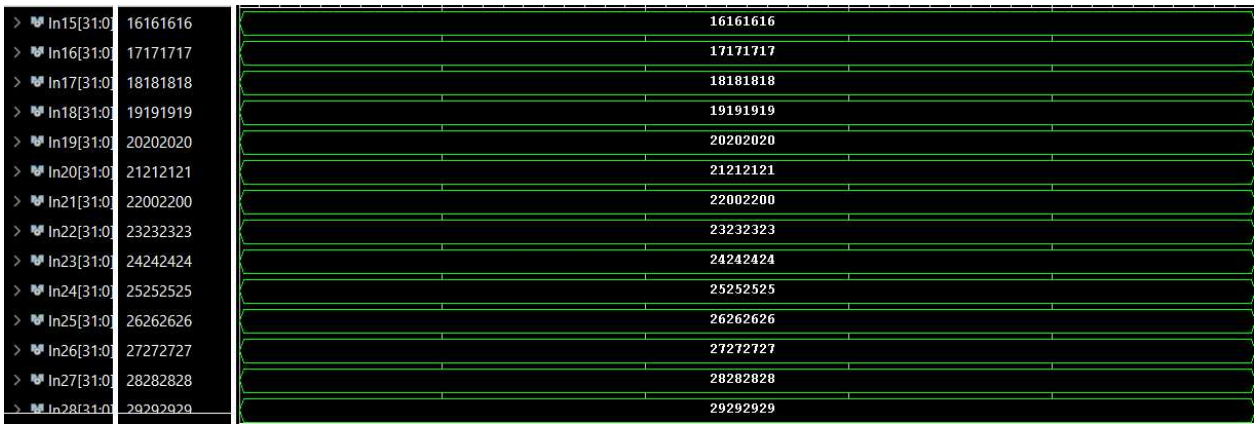

Q5 0
Q6 0
Q7 0
Q8 0
Q9 0
Q10 0
Q11 0
Q12 0
Q13 0
Q14 0
Q15 0

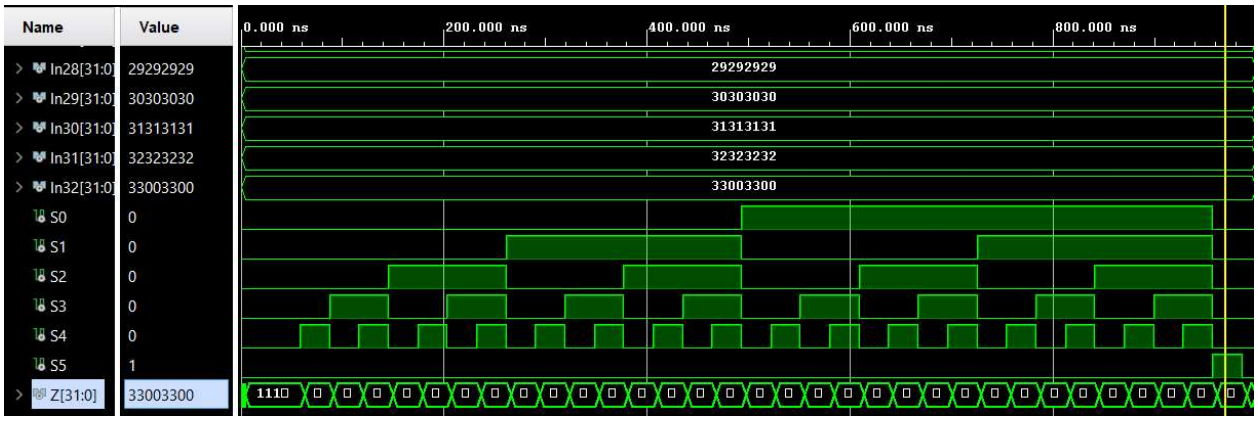

Q16 0
Q17 0
Q18 0
Q19 0
Q20 0
Q21 0
Q22 0
Q23 0
Q24 0


Q24 0
Q25 0
Q26 0
Q27 0
Q28 0
Q29 0
Q30 0
Q31 0
Q32 0

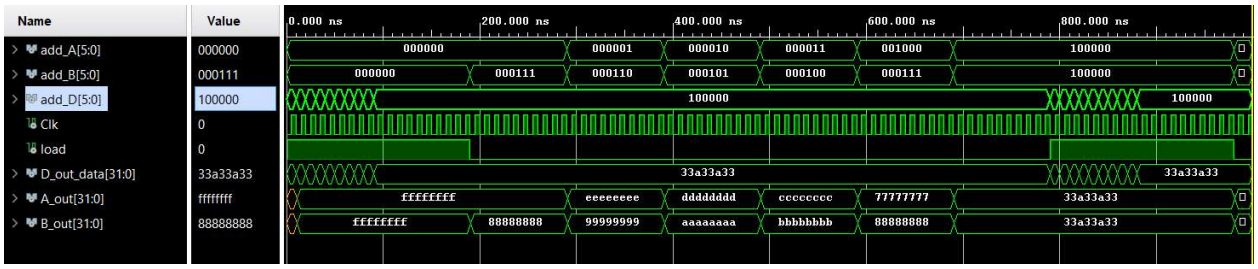
MUX 33 32 Bit:


In0[31:0] 11111111
In1[31:0] 22222222
In2[31:0] 33333333
In3[31:0] 44444444
In4[31:0] 55555555
In5[31:0] 66666666
In6[31:0] 77777777
In7[31:0] 88888888
In8[31:0] 99999999
In9[31:0] 10101010
In10[31:0] 11001100
In11[31:0] 12121212
In12[31:0] 13131313
In13[31:0] 14141414

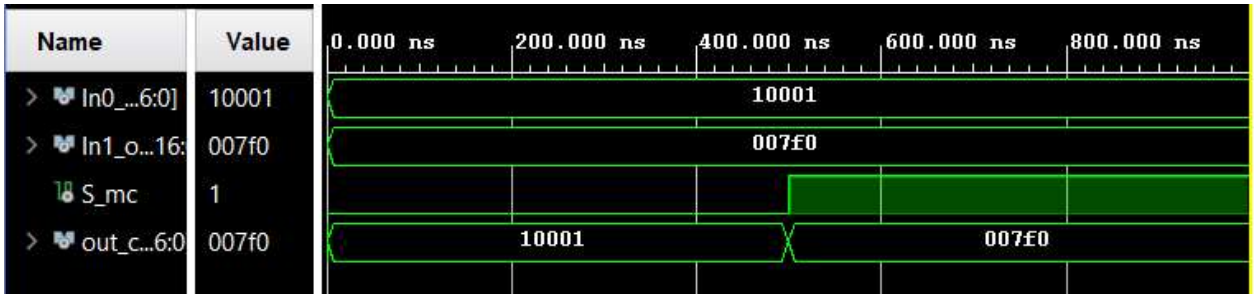

In15[31:0] 16161616
In16[31:0] 17171717
In17[31:0] 18181818
In18[31:0] 19191919
In19[31:0] 20202020
In20[31:0] 21212121
In21[31:0] 22002200
In22[31:0] 23232323
In23[31:0] 24242424
In24[31:0] 25252525
In25[31:0] 26262626
In26[31:0] 27272727
In27[31:0] 28282828
In28[31:0] 29292929


Name
Value
0.000 ns
200.000 ns
400.000 ns
600.000 ns
800.000 ns
In28[31:0] 29292929
In29[31:0] 30303030
In30[31:0] 31313131
In31[31:0] 32323232
In32[31:0] 33003300
S0 0
S1 0
S2 0
S3 0
S4 0
S5 1
Z[31:0] 33003300

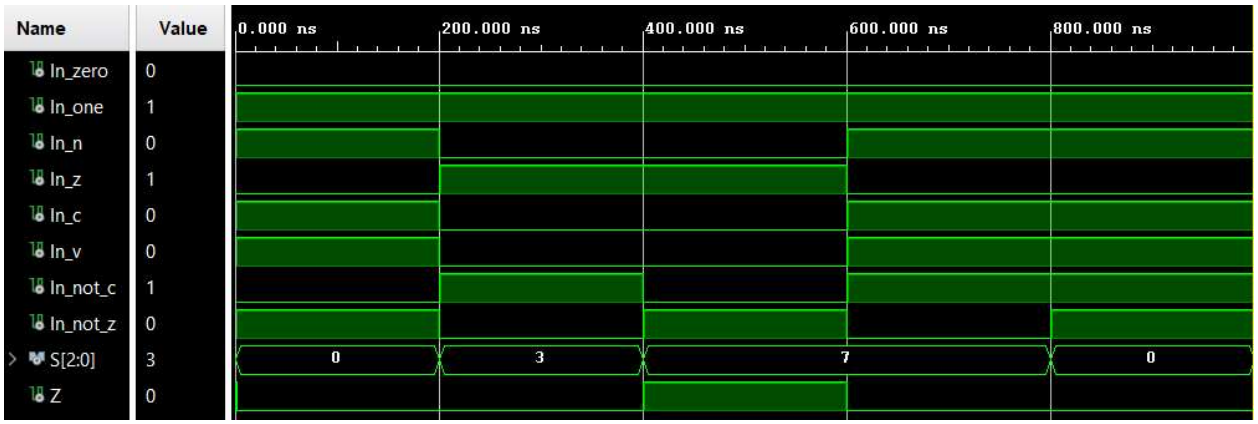
Register File with 33 Registers:



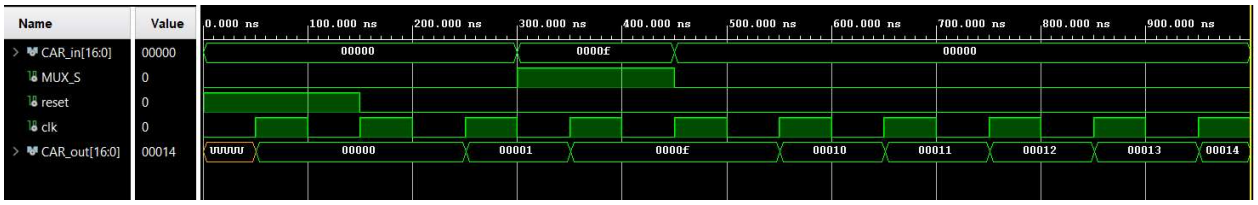
MUX 2 - 17 Bit:



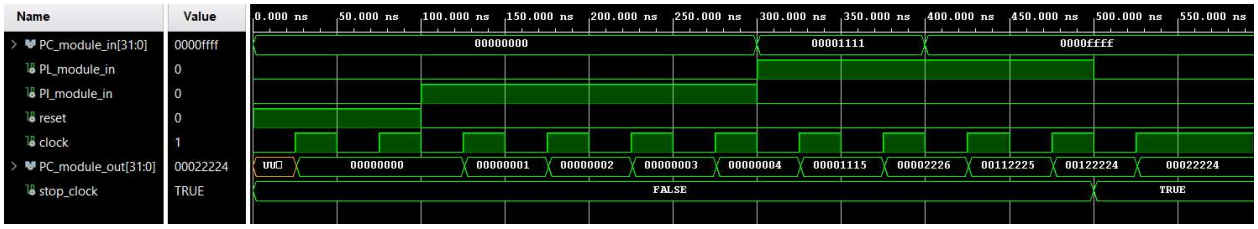
MUX 8 1 Bit:



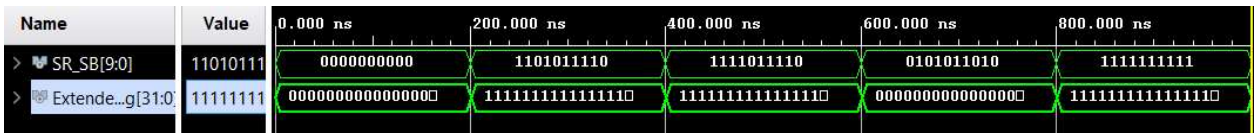
CAR:



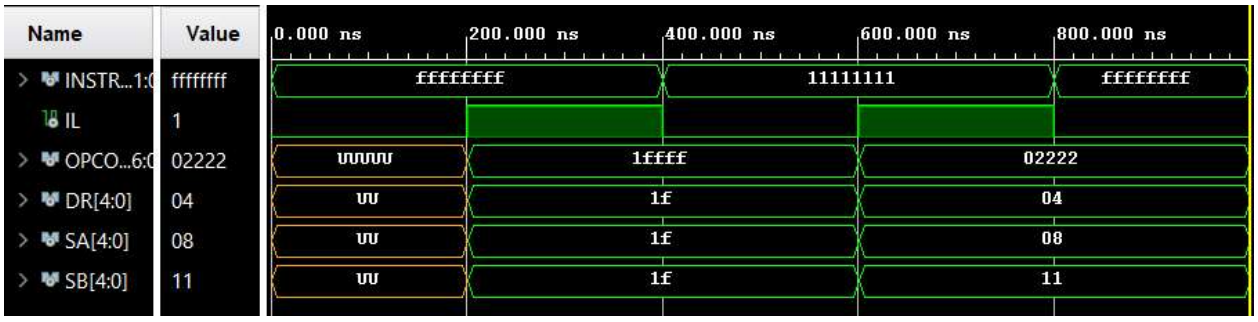
Programme Counter:



Extended PC:

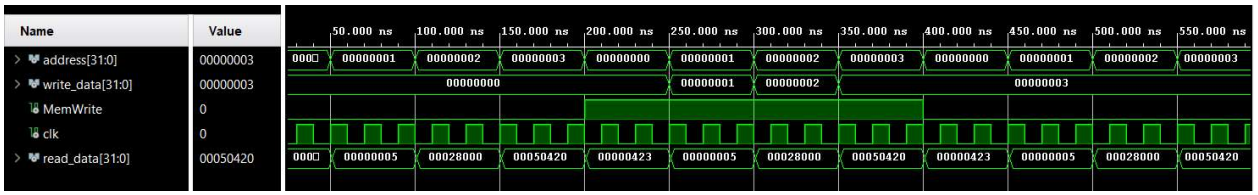


Instruction Register :

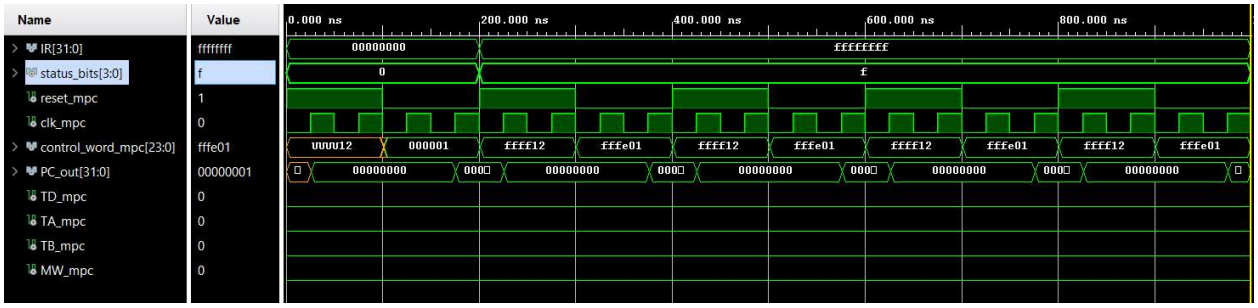


Control Memory :

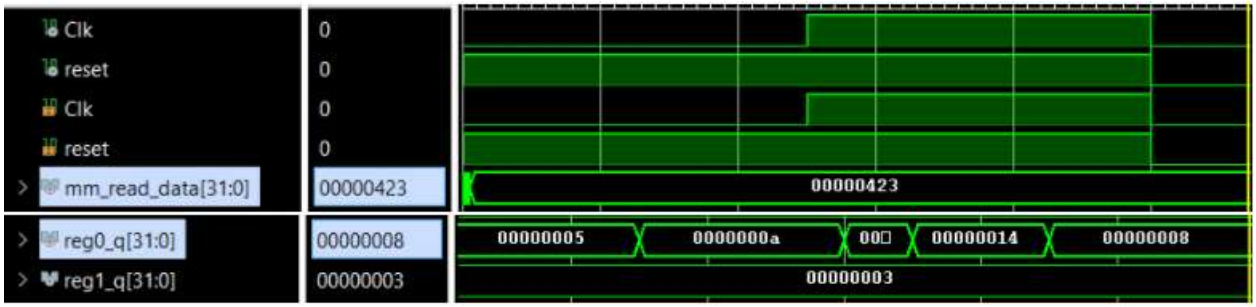
Memory :



Micro Programme Controller:



Processor:



THE END