Processor - Assignment 2 Report

Introduction:

For clarity reason I have decided to give an overview of my solution along with the actual VHDL code and test benches. These first two pages will provide a general overview of and the extent of completion of the assignment, then followed by the VHDL code itself.

For this assignment I had completed the previous two assignments which were giving to us. Therefore most of this existing code provided a good starting point for the processor. oI have already submitted both the two previous which included the smaller components and their test benches, therefore I felt it necessary to include them in this assignments.Below is a check list tasks giving to us by Michael Manzke and an indication as to its completion.

Check List:

-Increase the number of registers in the register-file from 8 to 9. Completed

-Add an additional select bit for the two multiplexers (Bus A and Bus
B) and the destination decoder. These are separate signals (TD, TA, TB) from the Control Memory. Completed

-Consequently adjust all components of the Datapath (e.g. MUXs in the register-file, decoder in the Register file, Arithmetic/logic Unit, Shifter and MUXs) are 16 bit operations. Completed

-Add and test Memory M (512 x 16) and Control Memory (256 x 28) to your project. MUX M will feed 16 bit addresses from ether the Bus A or the PC into the Memory M entity but only the 9 least significant address bits will be used to index into the array. This restricts the memory size to 512. Completed

-Implement all the components shown in Figure 1 on page 3. Completed

-Design reset logic for PC and CAR registers. This will enable you to start your program. Completed

-Write microprogramms for the Control Memory that implement the following instructions: ADI, LD, SR, INC, NOT, ADD, unconditional jump, and conditional branch (only one condition). Partially Completed

-Write machine code for the Memory M that demonstrates the use of the following instructions:
o ADI, LD, SR, INC, NOT, ADD, unconditional jump, and conditional branch (only one condition). Partially Completed

As you can see from the above I had was able to design the overall schematic of the processor and connect all its component in accordance, however I could only compete some of the machine code instructions. It was not a problem with my processor but however with the control memories implementation.

Microprogrammes Control:

All components were completed and connected accordingly

Datapath:

All components were completed and connected accordingly.

Memory:

All components were completed and connected accordingly.

Processor:

Unfortunately when connecting the three main sections of the processor I ran into some difficulties when trying to implement the micro-operations. Many of the signals coming out of the micro operational section would be undefined when it reached the data path, for example:

For a quite some time the DR and SA signals out of my Instruction register would be set correctly from the input from memory, however these DR and SA inputs would be undefined when reaching the register file.
Although this problem was fixed, it arose in many other occasions, and these undefined signals caused problems in the overall running of the processor.
I believe that this is just a signal mapping error and had I had a little bit more time, I could have fixed it and it would've worked with my already existing micro-operations.

Please find the VHDL code on the next page, accompanied with some test benches and screenshots.

VHDL Code

Processor:

```
entity Processor is
    Port ( CLK : in  STD_LOGIC;
           RESET : in  STD_LOGIC);
end Processor;

    architecture Behavioral of Processor is

component MicroprogrammedControl is
    Port ( Reset : in STD_LOGIC;
              oVerflow : in  STD_LOGIC;
           Carry : in  STD_LOGIC;
           Negative : in  STD_LOGIC;
           Zero : in  STD_LOGIC;
              CLK : in STD_LOGIC;
           IRInput : in  STD_LOGIC_VECTOR (15 downto 0);
           TD : out  STD_LOGIC;
           TA : out  STD_LOGIC;
           TB : out  STD_LOGIC;
              DR : out STD_LOGIC_VECTOR (2 downto 0);
              SA : out STD_LOGIC_VECTOR (2 downto 0);
              SB : out STD_LOGIC_VECTOR (2 downto 0);
           MB : out  STD_LOGIC;
           FS1 : out  STD_LOGIC_VECTOR(4 downto 0);
           MD : out  STD_LOGIC;
           RW : out  STD_LOGIC;
           MM : out  STD_LOGIC;
           MW : out  STD_LOGIC;

           PCout : out  STD_LOGIC_VECTOR (15 downto 0));
end component;

component Datapath is
    Port (
              PC : in STD_LOGIC_VECTOR(15 downto 0);
           DataIn : in  STD_LOGIC_VECTOR (15 downto 0);
              Selector: in STD_LOGIC_VECTOR (19 downto 0);
              DR : in STD_LOGIC_VECTOR (2 downto 0);
              SA : in STD_LOGIC_VECTOR (2 downto 0);
              SB : in STD_LOGIC_VECTOR (2 downto 0);
              fs : in STD_LOGIc_VECTOR (4 downto 0);
           AdressOut : out  STD_LOGIC_VECTOR (15 downto 0);
           Answer : out  STD_LOGIC_VECTOR (15 downto 0);
              MM : in STD_LOGIC;
              MD : in STD_LOGIC;
              MB : in STD_LOGIC;
              RW : in STD_LOGIC;
              reset : in STD_LOGIC;
              CLK : in  STD_LOGIC;
              V : OUT STD_LOGIC;
              C : OUT STD_LOGIC;
              N : OUT STD_LOGIC;
              Z : OUT STD_LOGIC;
           ConstantIn : in  STD_LOGIC_VECTOR (15 downto 0));

end component;
```

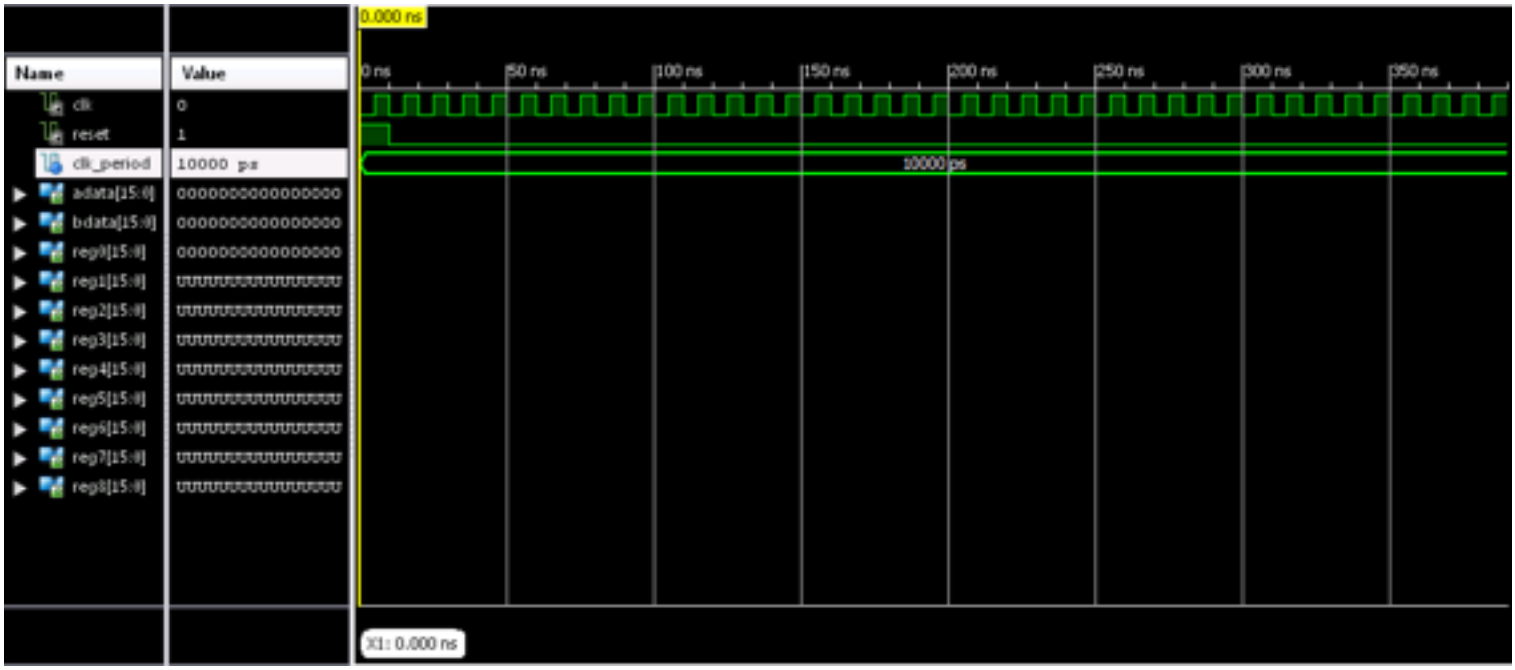
```
component  Memory_M is
    Port ( address :  in  STD_LOGIC_VECTOR (15 downto 0);
           write_data : in  STD_LOGIC_VECTOR (15 downto 0);
           MemWrite : in  STD_LOGIC;
           read_data : out  STD_LOGIC_VECTOR (15 downto 0));
end component;

signal memOutput, pcoutS, bSignal, aSignal, PCsignal : STD_LOGIC_VECTOR(15
downto 0);
signal mbS, mdS, rwS, mmS, mwS, Vs, Cs, Zs, mds1, taS, tdS, tbS, Ns : STD_LOGIC;
signal fs1S : std_logic_vector(4 downto 0);

signal saS, sbs, drS : STD_LOGIC_VECTOR(2 downto 0);
signal selectorS : STD_LOGIC_VECTOR(19 downto 0);
begin

--port maps
 microControl : MicroprogrammedControl PORT MAP(
        Reset => Reset,
        oVerflow => Vs,
      Carry => Cs,
      Negative => Ns,
      Zero => Zs,
        CLK => CLK,
      IRInput => memOutput,
      TD => tdS,
      TA => taS,
      TB => tbS,
        DR => drs,
       SA => sas,
        SB => sbs,
      MB => mbS,
      FS1 => fs1S,
      MD => mdS,
      RW => rwS,
      MM => mmS,
      MW => mwS,

      PCout => pcoutS

 );

--selectorS(19 downto 17) <= drS;
--selectorS(16 downto 14) <= saS;
--selectorS(13 downto 11) <= sbs;
--selectorS(7) <= mbS;
--selectorS(9 downto 5) <= fs1S;
--selectorS(1) <= mdS1;
selectorS(0) <= rwS;

 dPath : Datapath PORT MAP(
              PC => PCSignal,
              DataIn => memOutput,
              Selector => selectorS, ---tda + ta +tb etc.........
              DR => drs,
              SA => sas,
              SB => sbs,
              MB => mbs,
              MD => mds,
              FS => fs1s,
           AdressOut => aSignal,
           Answer => bSignal,
              MM => mmS,
              RW => rws,
              reset =>reset,
              CLK => CLK,
```

```
              V => vS,
              C => cS,
              N => nS,
              Z => zS,
           ConstantIn =>"0000000000000000"
);

 mem : Memory_M PORT MAP(
    address => aSignal,
   write_data => bSignal,
   MemWrite => selectorS(0),
   read_data => memOutput

 );


end Behavioral;
```

TESTBENCH:

```
ENTITY Processor_tb IS
END Processor_tb;

ARCHITECTURE behavior OF Processor_tb IS

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

    COMPONENT Processor
    PORT(
         CLK : IN  std_logic;
         RESET : IN  std_logic
        );
    END COMPONENT;


   --Inputs
   signal CLK : std_logic := '0';
   signal RESET : std_logic := '0';

   -- Clock period definitions
   constant CLK_period : time := 10 ns;

BEGIN

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

   -- Clock process definitions
   CLK_process :process
   begin
        CLK <= '0';
        wait for CLK_period/2;
        CLK <= '1';
        wait for CLK_period/2;
   end process;


   -- Stimulus process
```

```
   stim_proc: process
   begin
      -- hold reset state for 100 ns


        reset <= '1';
        wait for CLK_period;
        reset <= '0';
        wait for CLK_period;
        wait for CLK_period;
        wait for CLK_period;
        wait for CLK_period;
        wait for CLK_period;
        wait for CLK_period;
        wait for CLK_period;

      wait;
   end process;

END;
```

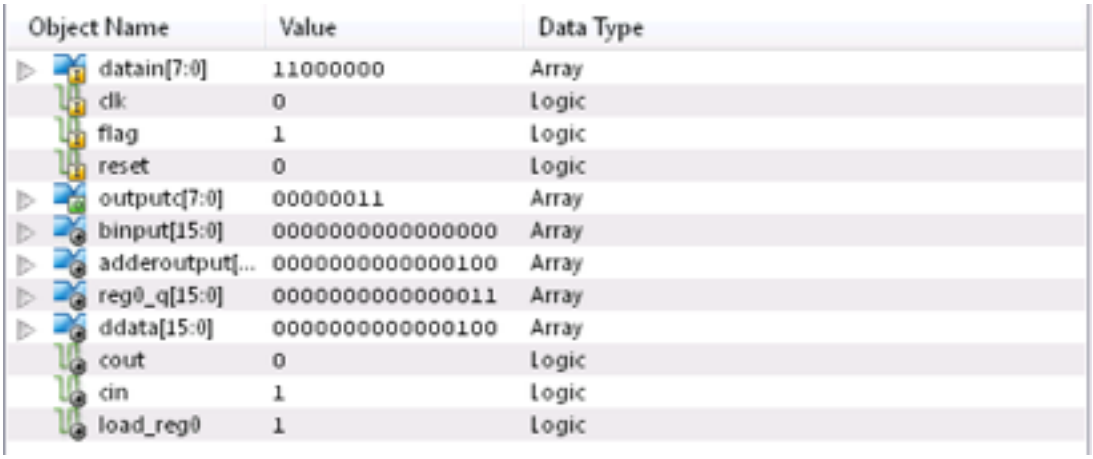


Microprocessor

CAR:

```
entity CAR is
    Port ( dataIn : in  STD_LOGIC_VECTOR (7 downto 0);
```

```
            CLK : in STD_LOGIC;
          flag : in  STD_LOGIC;
            reset : in STD_LOGIC;
            outputC : out STD_LOGIC_VECTOR (7 downto 0)
            );
end CAR;

architecture Behavioral of CAR is

COMPONENT Ripple_Adder is
    port( a : in  STD_LOGIC_VECTOR (15 downto 0);
          b : in  STD_LOGIC_VECTOR (15 downto 0);
          S : out  STD_LOGIC_VECTOR (15 downto 0);
          Cout : out  STD_LOGIC;
          Cin  : in  STD_LOGIC
         );
end component;

COMPONENT reg16
PORT(
        D : IN std_logic_vector(15 downto 0);
        load : IN std_logic;
        CLK : IN std_logic;
        Q : OUT std_logic_vector(15 downto 0)
    );
END COMPONENT;

signal  Binput, adderOutput, reg0_q, dData : STD_LOGIC_VECTOR(15 downto 0);
signal cout, Cin, load_reg0: STD_LOGIC;
begin

ripple_increment : Ripple_Adder PORT MAP(
    a => reg0_q,
    b => Binput,
    Cin => Cin,
    S => adderOutput,
    cout => cOut
    );

reg00: reg16 PORT MAP(
    D => Ddata,
    load => load_reg0,
    CLK => CLK,
    Q => reg0_q
    );

Binput <="0000000000000000";
Cin <= '1';
load_reg0 <= '1';
cOut <='0';

Ddata <= "0000000000000000" when(reset = '1') else
"00000000" &dataIn when(flag = '0') else adderoutput;


outputC <= reg0_q(7 downto 0);  --increment whats in reg




end Behavioral;
```



Object Name	Value	Data Type
datain[7:0]	11000000	Array
clk	0	Logic
flag	1	Logic
reset	0	Logic
outputc[7:0]	00000011	Array
binput[15:0]	0000000000000000	Array
adderoutput[...	0000000000000100	Array
reg0_q[15:0]	0000000000000011	Array
ddata[15:0]	0000000000000100	Array
cout	0	Logic
cin	1	Logic
load_reg0	1	Logic

Control Memory:

```
entity control_memory is
    Port ( IN_CAR : in  STD_LOGIC_VECTOR (7 downto 0);
           NA : out  STD_LOGIC_VECTOR (7 downto 0);
           MS1 : out  STD_LOGIC_VECTOR (2 downto 0);
           MC : out  STD_LOGIC;
           IL : out  STD_LOGIC;
           PL : out  STD_LOGIC;
              PI : out std_logic;
           TD : out  STD_LOGIC;
           TA : out  STD_LOGIC;
           TB : out  STD_LOGIC;
           MB : out  STD_LOGIC;
           FS1 : out  STD_LOGIC_VECTOR (4 downto 0);
           MD : out  STD_LOGIC;
           RW : out  STD_LOGIC;
              MW : out std_logic;
           MM : out  STD_LOGIC);
end control_memory;

architecture Behavioral of control_memory is

type mem_array is array(0 to 255) of std_logic_vector(27 downto 0);

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


 "11000000001000000010001001 00", --000 ADI
 "11000000001000000000000011 00", --001 LD
 "11000000001000000000000000 01", --002 ST
 "11000000001000000000000101 00", --003 INC
 "11000000001000000000111001 00", --004 NOT
 "11000000001000000000001100 00", --005 ADD
 "11000000001000100000000000 00", --006 --Branch nextAddress = execute
 "00000000000000000000000000 00", --007
 "00000000000000000000000000 00", --008
 "00000000000000000000000000 00", --009
 "00000000000000000000000000 00", --010

 "00000000000000000000000000 00", --011
 "00000000000000000000000000 00", --012
 "00000000000000000000000000 00", --013
 "00000000000000000000000000 00", --014
 "00000000000000000000000000 00", --015
 "00000000000000000000000000 00", --016
 "00000000000000000000000000 00", --017
 "00000000000000000000000000 00", --018
```

```
"00000000000000000000000000000000", --019
"00000000000000000000000000000000", --020

"00000000000000000000000000000000", --021
"00000000000000000000000000000000", --022
"00000000000000000000000000000000", --023
"00000000000000000000000000000000", --024
"00000000000000000000000000000000", --025
"00000000000000000000000000000000", --026
"00000000000000000000000000000000", --027
"00000000000000000000000000000000", --028
"00000000000000000000000000000000", --029
"00000000000000000000000000000000", --030

"00000000000000000000000000000000", --031
"00000000000000000000000000000000", --032
"00000000000000000000000000000000", --033
"00000000000000000000000000000000", --034
"00000000000000000000000000000000", --035
"00000000000000000000000000000000", --036
"00000000000000000000000000000000", --037
"00000000000000000000000000000000", --038
"00000000000000000000000000000000", --039
"00000000000000000000000000000000", --040

"00000000000000000000000000000000", --041
"00000000000000000000000000000000", --042
"00000000000000000000000000000000", --043
"00000000000000000000000000000000", --044
"00000000000000000000000000000000", --045
"00000000000000000000000000000000", --046
"00000000000000000000000000000000", --047
"00000000000000000000000000000000", --048
"00000000000000000000000000000000", --049
"00000000000000000000000000000000", --050

"00000000000000000000000000000000", --051
"00000000000000000000000000000000", --052
"00000000000000000000000000000000", --053
"00000000000000000000000000000000", --054
"00000000000000000000000000000000", --055
"00000000000000000000000000000000", --056
"00000000000000000000000000000000", --057
"00000000000000000000000000000000", --058
"00000000000000000000000000000000", --059
"00000000000000000000000000000000", --060

"00000000000000000000000000000000", --061
"00000000000000000000000000000000", --062
"00000000000000000000000000000000", --063
"00000000000000000000000000000000", --064
"00000000000000000000000000000000", --065
"00000000000000000000000000000000", --066
"00000000000000000000000000000000", --067
"00000000000000000000000000000000", --068
"00000000000000000000000000000000", --069
"00000000000000000000000000000000", --070

"00000000000000000000000000000000", --071
"00000000000000000000000000000000", --072
"00000000000000000000000000000000", --073
"00000000000000000000000000000000", --074
"00000000000000000000000000000000", --075
"00000000000000000000000000000000", --076
"00000000000000000000000000000000", --077
"00000000000000000000000000000000", --078
```

```
"00000000000000000000000000000000", --079
"00000000000000000000000000000000", --080

"00000000000000000000000000000000", --081
"00000000000000000000000000000000", --082
"00000000000000000000000000000000", --083
"00000000000000000000000000000000", --084
"00000000000000000000000000000000", --085
"00000000000000000000000000000000", --086
"00000000000000000000000000000000", --087
"00000000000000000000000000000000", --088
"00000000000000000000000000000000", --089

"00000000000000000000000000000000", --090
"00000000000000000000000000000000", --091
"00000000000000000000000000000000", --092
"00000000000000000000000000000000", --093
"00000000000000000000000000000000", --094
"00000000000000000000000000000000", --095
"00000000000000000000000000000000", --096
"00000000000000000000000000000000", --097
"00000000000000000000000000000000", --098

"00000000000000000000000000000000", --099
"00000000000000000000000000000000", --100
"00000000000000000000000000000000", --101
"00000000000000000000000000000000", --102
"00000000000000000000000000000000", --103
"00000000000000000000000000000000", --104
"00000000000000000000000000000000", --105
"00000000000000000000000000000000", --106
"00000000000000000000000000000000", --107
"00000000000000000000000000000000", --108
"00000000000000000000000000000000", --109
"00000000000000000000000000000000", --110

"00000000000000000000000000000000", --111
"00000000000000000000000000000000", --112
"00000000000000000000000000000000", --113
"00000000000000000000000000000000", --114
"00000000000000000000000000000000", --115
"00000000000000000000000000000000", --116
"00000000000000000000000000000000", --117
"00000000000000000000000000000000", --118
"00000000000000000000000000000000", --119
"00000000000000000000000000000000", --120

"00000000000000000000000000000000", --121
"00000000000000000000000000000000", --122
"00000000000000000000000000000000", --123
"00000000000000000000000000000000", --124
"00000000000000000000000000000000", --125
"00000000000000000000000000000000", --126
"00000000000000000000000000000000", --127
"00000000000000000000000000000000", --128
"00000000000000000000000000000000", --129
"00000000000000000000000000000000", --130

"00000000000000000000000000000000", --131
"00000000000000000000000000000000", --132
"00000000000000000000000000000000", --133
"00000000000000000000000000000000", --134
"00000000000000000000000000000000", --135
"00000000000000000000000000000000", --136
"00000000000000000000000000000000", --137
"00000000000000000000000000000000", --138
```

```
"00000000000000000000000000000000", --139

"00000000000000000000000000000000", --140
"00000000000000000000000000000000", --141
"00000000000000000000000000000000", --142
"00000000000000000000000000000000", --143
"00000000000000000000000000000000", --144
"00000000000000000000000000000000", --145
"00000000000000000000000000000000", --146
"00000000000000000000000000000000", --147
"00000000000000000000000000000000", --148
"00000000000000000000000000000000", --149


"00000000000000000000000000000000", --150
"00000000000000000000000000000000", --151
"00000000000000000000000000000000", --152
"00000000000000000000000000000000", --153
"00000000000000000000000000000000", --154
"00000000000000000000000000000000", --155
"00000000000000000000000000000000", --156
"00000000000000000000000000000000", --157
"00000000000000000000000000000000", --158
"00000000000000000000000000000000", --159

"00000000000000000000000000000000", --160
"00000000000000000000000000000000", --161
"00000000000000000000000000000000", --162
"00000000000000000000000000000000", --163
"00000000000000000000000000000000", --164
"00000000000000000000000000000000", --165
"00000000000000000000000000000000", --166
"00000000000000000000000000000000", --167
"00000000000000000000000000000000", --168
"00000000000000000000000000000000", --169

"00000000000000000000000000000000", --170
"00000000000000000000000000000000", --171
"00000000000000000000000000000000", --172
"00000000000000000000000000000000", --173
"00000000000000000000000000000000", --174
"00000000000000000000000000000000", --175
"00000000000000000000000000000000", --176
"00000000000000000000000000000000", --177
"00000000000000000000000000000000", --178
"00000000000000000000000000000000", --179

"00000000000000000000000000000000", --180
"00000000000000000000000000000000", --181
"00000000000000000000000000000000", --182
"00000000000000000000000000000000", --183
"00000000000000000000000000000000", --184
"00000000000000000000000000000000", --185
"00000000000000000000000000000000", --186
"00000000000000000000000000000000", --187
"00000000000000000000000000000000", --188
"00000000000000000000000000000000", --189

"00000000000000000000000000000000", --190
"00000000000000000000000000000000", --191
"11000001000011000000000000010", --192 fetch
"00000000001100000000000000000", --193 execute
"00000000000000000000000000000000", --194
"00000000000000000000000000000000", --195
"00000000000000000000000000000000", --196
"00000000000000000000000000000000", --197
```

```
"00000000000000000000000000000000", --198
"00000000000000000000000000000000", --199

"00000000000000000000000000000000", --200
"00000000000000000000000000000000", --201
"00000000000000000000000000000000", --202
"00000000000000000000000000000000", --203
"00000000000000000000000000000000", --204
"00000000000000000000000000000000", --205
"00000000000000000000000000000000", --206
"00000000000000000000000000000000", --207
"00000000000000000000000000000000", --208
"00000000000000000000000000000000", --209

"00000000000000000000000000000000", --210
"00000000000000000000000000000000", --211
"00000000000000000000000000000000", --212
"00000000000000000000000000000000", --213
"00000000000000000000000000000000", --214
"00000000000000000000000000000000", --215
"00000000000000000000000000000000", --216
"00000000000000000000000000000000", --217
"00000000000000000000000000000000", --218
"00000000000000000000000000000000", --219

"00000000000000000000000000000000", --220
"00000000000000000000000000000000", --221
"00000000000000000000000000000000", --222
"00000000000000000000000000000000", --223
"00000000000000000000000000000000", --224
"00000000000000000000000000000000", --225
"00000000000000000000000000000000", --226
"00000000000000000000000000000000", --227
"00000000000000000000000000000000", --228
"00000000000000000000000000000000", --229

"00000000000000000000000000000000", --230
"00000000000000000000000000000000", --232
"00000000000000000000000000000000", --232
"00000000000000000000000000000000", --233
"00000000000000000000000000000000", --234
"00000000000000000000000000000000", --235
"00000000000000000000000000000000", --236
"00000000000000000000000000000000", --237
"00000000000000000000000000000000", --238
"00000000000000000000000000000000", --239

"00000000000000000000000000000000", --240
"00000000000000000000000000000000", --241
"00000000000000000000000000000000", --242
"00000000000000000000000000000000", --243
"00000000000000000000000000000000", --244
"00000000000000000000000000000000", --245
"00000000000000000000000000000000", --246
"00000000000000000000000000000000", --247
"00000000000000000000000000000000", --248
"00000000000000000000000000000000", --249

"00000000000000000000000000000000", --250
"00000000000000000000000000000000", --251
"00000000000000000000000000000000", --252
"00000000000000000000000000000000", --253
"00000000000000000000000000000000", --254
"00000000000000000000000000000000" --255
```

```
);
variable addr : integer;
variable control_out : std_logic_vector(27 downto 0);

begin

addr := conv_integer(IN_CAR);
control_out := control_mem(addr);
MW <= control_out(0);
MM <= control_out(1);
RW <= control_out(2);
MD <= control_out(3);
FS1 <= control_out(8 downto 4); MB <= control_out(9);
TB <= control_out(10);
TA <= control_out(11);
TD <= control_out(12);
PL <= control_out(13);
PI <= control_out(14);
IL <= control_out(15);
MC <= control_out(16);
MS1 <= control_out(19 downto 17); NA <= control_out(27 downto 20); end process;


end Behavioral;
```

Extend:

```
entity extend is
    Port ( DR : in  STD_LOGIC_VECTOR (2 downto 0);
           SB : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : out  STD_LOGIC_VECTOR (15 downto 0));
end extend;

architecture Behavioral of extend is


begin

    Z(15 downto 6) <= "0000000000" when(DR(2) ='0') else
    "1111111111";
    Z(5 downto 3) <= DR;
    Z(2 downto 0) <= SB;




end Behavioral;
```

Instruction Register:

```
entity InstructionRegister is
    Port ( dataIn : in  STD_LOGIC_VECTOR (15 downto 0);
           IL : in  STD_LOGIC;
```

```
            opCode : out  STD_LOGIC_VECTOR (6 downto 0);
               CLK : in STD_LOGIC;
            DR : out  STD_LOGIC_VECTOR (2 downto 0);
            SA : out  STD_LOGIC_VECTOR (2 downto 0);
            SB : out  STD_LOGIC_VECTOR (2 downto 0));
end InstructionRegister;


architecture Behavioral of InstructionRegister is




component reg16
port(
    D : IN std_logic_vector(15 downto 0);
    load : IN std_logic;
    CLK : IN std_logic;
    Q : OUT std_logic_vector(15 downto 0)
    );
END COMPONENT;

begin


    reg: reg16 port map
    (
        D => DataIN,
        load => not(IL),
        CLK => CLK,
        Q(15 downto 9)=> opcode,
        Q(8 downto 6) => DR,
        Q(5 downto 3) =>SA,
        Q(2 downto 0) => SB
    );




end Behavioral;
```

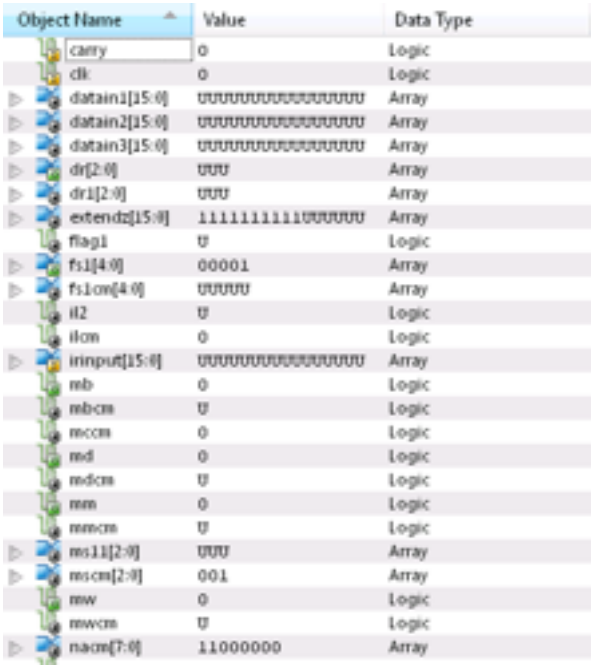
Microprogrammed Control:

```
entity MicroprogrammedControl is
    Port ( Reset : in STD_LOGIC;
               oVerflow : in  STD_LOGIC;
           Carry : in  STD_LOGIC;
           Negative : in  STD_LOGIC;
           Zero : in  STD_LOGIC;
               CLK : in STD_LOGIC;
           IRInput : in  STD_LOGIC_VECTOR (15 downto 0);
           TD : out  STD_LOGIC;
           TA : out  STD_LOGIC;
           TB : out  STD_LOGIC;
               DR : out STD_LOGIC_VECTOR (2 downto 0);
               SA : out STD_LOGIC_VECTOR (2 downto 0);
               SB : out STD_LOGIC_VECTOR (2 downto 0);
           MB : out  STD_LOGIC;
           FS1 : out  STD_LOGIC_VECTOR(4 downto 0);
           MD : out  STD_LOGIC;
```

```
           RW : out  STD_LOGIC;
           MM : out  STD_LOGIC;
           MW : out  STD_LOGIC;

           PCout : out  STD_LOGIC_VECTOR (15 downto 0));
end MicroprogrammedControl;

architecture Behavioral of MicroprogrammedControl is



COMPONENT PC is
    Port ( DataIn : in  STD_LOGIC_VECTOR (15 downto 0);
           reset : in STD_LOGIC;
              CLK : in STD_LOGIC;
           PL : in  STD_LOGIC;
           PI : in  STD_LOGIC;
           Z : out  STD_LOGIC_VECTOR (15 downto 0));
end COMPONENT;

component  InstructionRegister is
    Port ( dataIn : in  STD_LOGIC_VECTOR (15 downto 0);
           IL : in  STD_LOGIC;
           opCode : out  STD_LOGIC_VECTOR (6 downto 0);
              CLK : in STD_LOGIC;
           DR : out  STD_LOGIC_VECTOR (2 downto 0);
           SA : out  STD_LOGIC_VECTOR (2 downto 0);
           SB : out  STD_LOGIC_VECTOR (2 downto 0));
end component;

component CAR is
    Port ( dataIn : in  STD_LOGIC_VECTOR (7 downto 0);
              CLK : in STD_LOGIC;
              reset : in STD_LOGIC;
           flag : in  STD_LOGIC;
              outputC : out STD_LOGIC_VECTOR (7 downto 0)
              );
end component;

component control_memory is
    Port ( IN_CAR : in  STD_LOGIC_VECTOR (7 downto 0);
           NA : out  STD_LOGIC_VECTOR (7 downto 0);
           MS1 : out  STD_LOGIC_VECTOR (2 downto 0);
           MC : out  STD_LOGIC;
           IL : out  STD_LOGIC;
           PL : out  STD_LOGIC;
              PI : out std_logic;
           TD : out  STD_LOGIC;
           TA : out  STD_LOGIC;
           TB : out  STD_LOGIC;
           MB : out  STD_LOGIC;
           FS1 : out  STD_LOGIC_VECTOR (4 downto 0);
           MD : out  STD_LOGIC;
           RW : out  STD_LOGIC;
              MW : out std_logic;
           MM : out  STD_LOGIC);
end component;

component MUX_S is
    Port ( In0 : in  STD_LOGIC;
           In1 : in  STD_LOGIC;
           C : in  STD_LOGIC;
           V : in  STD_LOGIC;
           N : in  STD_LOGIC;
           Z : in  STD_LOGIC;
           notC : in  STD_LOGIC;
```

```
            notZ : in  STD_LOGIC;
            MS1 : in  STD_LOGIC_VECTOR (2 downto 0);
            output : out  STD_LOGIC);
end component;

component MUX_C is
    Port ( NA : in  STD_LOGIC_VECTOR (7 downto 0);
           opcode : in  STD_LOGIC_VECTOR (6 downto 0);
           MC : in  STD_LOGIC;
           output : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component extend is
    PORT (  DR : in  STD_LOGIC_VECTOR (2 downto 0);
            SB : in  STD_LOGIC_VECTOR (2 downto 0);
            Z : out  STD_LOGIC_VECTOR (15 downto 0));

end component;

signal dataIn1, Zpc, dataIn2, dataIn3 , extendZ : STD_LOGIC_VECTOR(15 downto 0);
signal opcode2 : STD_LOGIC_VECTOR(6 downto 0);
signal DR1, SA1, SB1, MS11, MScm : STD_LOGIC_VECTOR(2 downto 0);
signal FS1cm : STD_LOGIC_VECTOR(4 downto 0);
signal outPutC, NAcm, outputMUX_C, outputC1 : STD_LOGIC_VECTOR(7 downto 0);
signal PL1, PI1, reset1, IL2, flag1, MCcm, ILcm, PLcm, PIcm, TDcm, TAcm, TBcm,
MBcm, MDcm, RWcm, MWcm, MMcm, notCarry, notZero, outputMUX_S  : STD_LOGIC;

begin

--port maps

    mPC : PC PORT MAP(
        DataIn => extendZ,
      reset => reset,
        CLK => CLK,
      PL => PLcm,
      PI => PIcm,
      Z => Zpc

    );

    mIR : InstructionRegister PORT MAP(
        dataIn => IRInput,
      IL => ILcm,
      opCode => opcode2,
        CLK => CLK,
      DR => DR,
      SA => SA,
      SB => SB
        );


    mCar : CAR PORT MAP(
        dataIN => outputMUX_C,
        CLK => CLK,
        reset=>reset,
      flag => outputMUX_S,
        outputC => outputC1
    );

    mControlMemory : Control_memory PORT MAP(
        IN_CAR => outputC1,
      NA => NAcm,
      MS1 => MScm,
      MC => MCcm,
      IL => ILcm,
```

```
      PL => PLcm,
        PI => PIcm,
      TD => TDcm,
      TA => TAcm,
      TB => TBcm,
      MB => MB,
      FS1 => FS1,
      MD => MD,
      RW => RW,
        MW => MW,
      MM => MM
    );

    notCarry <= not(Carry);
    notZero <= not(Zero);

    mMUX_S : MUX_S PORT MAP(
        In0 => '0',
      In1 => '1',
      C => Carry,
      V => overFlow,
      N => negative,
      Z => Zero,
      notC => notCarry,
      notZ => notZero,
      MS1 => MScm,
      output => outputMUX_S

    );

    mMUX_C : MUX_C PORT MAP(
        NA => NAcm,
      opcode => opcode2,
      MC => MCcm,
      output => outputMUX_C

    );

    extendM : extend PORT MAP(
        DR => DR1,
        SB => SB1,
        Z => extendZ
    );

end behavioral;
```



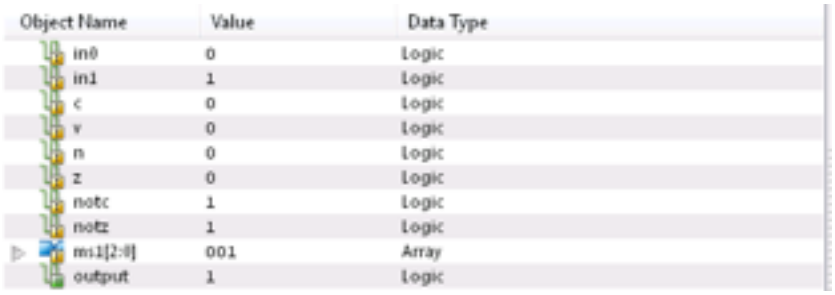
MUX_S:

```
entity MUX_S is
    Port ( In0 : in  STD_LOGIC;
           In1 : in  STD_LOGIC;
           C : in  STD_LOGIC;
           V : in  STD_LOGIC;
           N : in  STD_LOGIC;
           Z : in  STD_LOGIC;
           notC : in  STD_LOGIC;
           notZ : in  STD_LOGIC;
           MS1 : in  STD_LOGIC_VECTOR (2 downto 0);
           output : out  STD_LOGIC);
end MUX_S;

architecture Behavioral of MUX_S is

begin

output <= '0' after 1 ns when(MS1 = "000") else
'1' after 1 ns when(MS1 = "001") else
C after 1 ns when(MS1 = "010") else
V after 1 ns when(MS1 = "011") else
N after 1 ns when(MS1 = "100") else
Z after 1 ns when(MS1 = "101") else
notC after 1 ns when(MS1 = "110") else
notZ after 1 ns when(MS1 = "111");


end Behavioral;
```

PC:

```
entity PC is
    Port ( DataIn : in  STD_LOGIC_VECTOR (15 downto 0);
              reset : in STD_LOGIC;
              CLK : in STD_LOGIC;
           PL : in  STD_LOGIC;
           PI : in  STD_LOGIC;
           Z : out  STD_LOGIC_VECTOR (15 downto 0));
end PC;




architecture Behavioral of PC is

COMPONENT reg16
PORT(
        D : IN std_logic_vector(15 downto 0);
        load : IN std_logic;
        CLK : IN std_logic;
        Q : OUT std_logic_vector(15 downto 0)
    );
END COMPONENT;

COMPONENT Ripple_Adder is
    port( a : in  STD_LOGIC_VECTOR (15 downto 0);
         b : in  STD_LOGIC_VECTOR (15 downto 0);
```

```
          S : out  STD_LOGIC_VECTOR (15 downto 0);
          Cout : out  STD_LOGIC;
          Cin  : in  STD_LOGIC
        );
end component;

signal realInput, regInput, adderInput, adderOutput, reg0_q, DSM, b0 :
STD_LOGIC_VECTOR(15 downto 0);
signal cOUt, c0 : STD_LOGIC;

begin



reg00: reg16 PORT MAP(
    D => DSM,
    load => '1',
    CLK => CLK,
    Q => reg0_q
    );

ripple_increment : Ripple_Adder PORT MAP(
    a => reg0_q,
    b => b0,
    Cin => c0,
    S => adderOutput,
    cout => cOut
    );

    b0 <= "0000000000000000" when(pi <= '1') or (reset <= '1')
    else realInput when(pl = '1');
    c0 <= '1' when(pi = '1') else '0';

    DSM <= "0000000000000000" when(reset = '1')
    else adderOutPut;

    Z <= reg0_q;

                                                    end Behavioral;
```

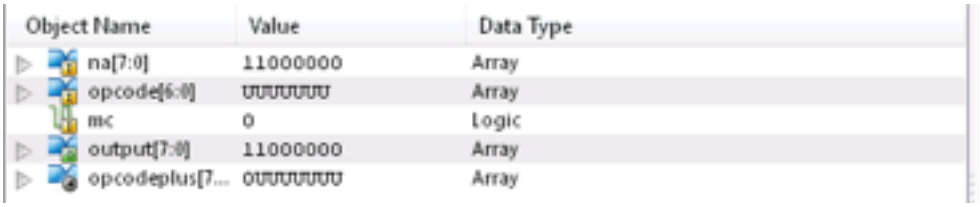


Object Name	Value	Data Type
in0	0	Logic
in1	1	Logic
c	0	Logic
v	0	Logic
n	0	Logic
z	0	Logic
notc	1	Logic
notz	1	Logic
ms1[2:0]	001	Array
output	1	Logic

MUX_C:

```
entity MUX_C is
    Port ( NA : in  STD_LOGIC_VECTOR (7 downto 0);
           opcode : in  STD_LOGIC_VECTOR (6 downto 0);
           MC : in  STD_LOGIC;
           output : out  STD_LOGIC_VECTOR (7 downto 0));
end MUX_C;


architecture Behavioral of MUX_C is
```

```
signal opcodePluS : std_logic_vector (7 downto 0);

begin

opcodePlus(7) <=  MC;
opcodePlus(6 downto 0) <= opcode;

output <= NA after 5 ns when(MC = '0')  else  --nextAddress
opcodePlus after 5ns when (MC ='1');


end Behavioral;
```



DataPath

Functional Unit:

```
entity FunctionalUnit is
    Port ( fuA : in  STD_LOGIC_VECTOR (15 downto 0);
           fuB : in  STD_LOGIC_VECTOR (15 downto 0);
           fuZ : out  STD_LOGIC_VECTOR (15 downto 0);
              FSelect: in STD_LOGIC_VECTOR (4 downto 0);
           oVerflow : out  STD_LOGIC;
           Carry : out  STD_LOGIC;
           Zero : out  STD_LOGIC;
           Negative : out  STD_LOGIC);
end FunctionalUnit;

architecture Behavioral of FunctionalUnit is

COMPONENT ALU
    port(   S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           S2 : in  STD_LOGIC;
           S3 : in  STD_LOGIC;
           Z : out  STD_LOGIC_VECTOR (15 downto 0);
              CarryOut : out STD_LOGIC;
              overFlow : out STD_logic;
              Zero     : out Std_logic;
              negative : out std_logic;
           InAlu0 : in  STD_LOGIC_VECTOR (15 downto 0);
           InAlu1 : in  STD_LOGIC_VECTOR (15 downto 0));
end component;

COMPONENT Sixteen_bit_Shifter
    port(  A : in  STD_LOGIC_VECTOR (15 downto 0);
```

```
            Z : out  STD_LOGIC_VECTOR (15 downto 0);
            H0 : in  STD_LOGIC;
               H1 : in  STD_LOGIC);
end component;
signal z0, z1  : std_logic_vector(15 downto 0);
signal N, C, overF, nega, zer : std_logic;
begin

-- port maps ;-)
-- ALU
    ALU_1: ALU PORT MAP(
    S0 => FSelect(3),
    S1 => FSelect(2),
    S2 => FSelect(1),
    S3 => FSelect(0),
    Z => z0,
    CarryOut => c,
    overFlow => overF,
    Zero => zer,
    negative => nega,
    InAlu0 => FuA,
    InAlu1 => FuB
    );

    -- port maps ;-)
-- Shifter
    Shifter: Sixteen_bit_Shifter PORT MAP(
   H0 => FSelect(2),
    H1 => FSelect(3),
    A => FuB,
    Z => Z1
    );

 Negative <= nega ;
 Carry <= c;
 Zero <= zer;
 oVerflow <=overF ;



 fuZ <= z0 when(FSelect(4) = '0') else Z1;




end Behavioral;
```

Object Name	Value	Data Type
fua[15:0]	0000000000000000	Array
fub[15:0]	0000000000000000	Array
fuz[15:0]	0000000000000001	Array
fselect[4:0]	00001	Array
overflow	0	Logic
carry	0	Logic
zero	0	Logic
negative	0	Logic
z0[15:0]	0000000000000001	Array
z1[15:0]	0000000000000000	Array
n	U	Logic
c	0	Logic
overf	0	Logic
nega	0	Logic
zer	0	Logic

MUX_M:

```
entity mux2to16 is
    Port ( s : in  STD_LOGIC;
           In0 : in  STD_LOGIC_VECTOR (15 downto 0);
           In1 : in  STD_LOGIC_VECTOR (15 downto 0);
           Z : out  STD_LOGIC_VECTOR (15 downto 0));
end mux2to16;

architecture Behavioral of mux2to16 is

begin

Z <= In0 after 5 ns when S='0' else
In1 after 5 ns when S='1'else
"0000000000000000" after 5 ns;

end Behavioral;
```

RegFile:

```
entity RegFile2 is
    Port ( DData : in  STD_LOGIC_VECTOR (15 downto 0);
              reset : in STD_LOGIC;
           DSelect : in  STD_LOGIC_VECTOR (2 downto 0);
           ASelect  : in  STD_LOGIC_VECTOR (2 downto 0);
              BSelect  : in  STD_LOGIC_VECTOR (2 downto 0);
           AData : out  STD_LOGIC_VECTOR (15 downto 0);
           BData : out  STD_LOGIC_VECTOR (15 downto 0);
              reg0 : out  STD_LOGIC_VECTOR (15 downto 0);
              CLK : in STD_LOGIC;
           reg1 : out  STD_LOGIC_VECTOR (15 downto 0);
           reg2 : out  STD_LOGIC_VECTOR (15 downto 0);
           reg3 : out  STD_LOGIC_VECTOR (15 downto 0);
           reg4 : out  STD_LOGIC_VECTOR (15 downto 0);
           reg5 : out  STD_LOGIC_VECTOR (15 downto 0);
           reg6 : out  STD_LOGIC_VECTOR (15 downto 0);
           reg7 : out  STD_LOGIC_VECTOR (15 downto 0);
              reg8 : out STD_LOGIC_VECTOR  (15 downto 0)
              );
end RegFile2;

architecture Behavioral of RegFile2 is

-- 4 bit Register for register file

COMPONENT reg16
PORT(
        D : IN std_logic_vector(15 downto 0);
        load : IN std_logic;
        CLK : IN std_logic;
        Q : OUT std_logic_vector(15 downto 0)
    );
END COMPONENT;
```

```
coMPONENT mux_2to8
PORT (      S0 : in  STD_LOGIC;
            S1 : in  STD_LOGIC;
            S2 : in  STD_LOGIC;

            In0 : in  STD_LOGIC_VECTOR (15 downto 0);
            In1 : in  STD_LOGIC_VECTOR (15 downto 0);
            In2 : in  STD_LOGIC_VECTOR (15 downto 0);
            In3 : in  STD_LOGIC_VECTOR (15 downto 0);
            In4 : in  STD_LOGIC_VECTOR (15 downto 0);
            In5 : in  STD_LOGIC_VECTOR (15 downto 0);
            In6 : in  STD_LOGIC_VECTOR (15 downto 0);
            In7 : in  STD_LOGIC_VECTOR (15 downto 0);
               In8 : in STD_LOGIC_VECTOR (15 downto 0);
            Z : out  STD_LOGIC_VECTOR (15 downto 0));
END COMPONENT;

-- 2+1 to 4X2 Decoder
COMPONENT decoder_3to8
PORT(
    A0 : IN std_logic;
    A1 : IN std_logic;
    A2 : 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
    );
END COMPONENT;



-- signals
signal load_reg0, load_reg1, load_reg2, load_reg3, load_reg4, load_reg5,
load_reg6, load_reg7, load_reg8 : std_logic;
signal reg0_q, reg1_q, reg2_q, reg3_q, reg4_q, reg5_q, reg6_q, reg7_q, reg8_q,

data_src_mux_out, src_reg, output0, output1 : std_logic_vector(15 downto 0);

begin

-- port maps ;-)
-- register 0
    reg00: reg16 PORT MAP(
    D => Ddata,
    load => load_reg0,
    CLK => CLK,
    Q => reg0_q
    );
-- register 1
    reg01: reg16 PORT MAP(
    D => Ddata,
    load => load_reg1,
    CLK => CLK,
    Q => reg1_q
    );

-- register 2
```

```
    reg02: reg16 PORT MAP(
    D => Ddata,
    load => load_reg2,
    CLK => CLK,
    Q => reg2_q
    );

-- register 3
    reg03: reg16 PORT MAP(
    D => Ddata,
    load => load_reg3,
    CLK => CLK,
    Q => reg3_q
    );

-- register 4
    reg04: reg16 PORT MAP(
    D => Ddata,
    load => load_reg4,
    CLK => CLK,
    Q => reg4_q
    );

-- register 5
    reg05: reg16 PORT MAP(
    D => Ddata,
    load => load_reg5,
    CLK => CLK,
    Q => reg5_q
    );

-- register 6
    reg06: reg16 PORT MAP(
    D => Ddata,
    load => load_reg6,
    CLK => CLK,
    Q => reg6_q
    );

-- register 7
    reg07: reg16 PORT MAP(
    D => Ddata,
    load => load_reg7,
    CLK => CLK,
    Q => reg7_q
    );

    -- register 8
    reg08: reg16 PORT MAP(
    D => Ddata,
    load => load_reg8,
    CLK => CLK,
    Q => reg8_q
    );


-- port maps ;-)
-- mUX_1

    MUX_1 : mux_2to8 PORT MAP(

             S0 => ASelect(0),
            S1 => ASelect(1),
            S2 => Aselect(2),

            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,
               z => AData
    );

    MUX_2 : mux_2to8 PORT MAP(

            S0 => BSelect(0),
            S1 => BSelect(1),
            S2 => Bselect(2),

            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,
               z => Bdata

    );

-- sOURCE register decoder
    des_decoder :decoder_3to8 PORT MAP(
    A0 =>  DSelect(2),
    A1 =>  DSelect(1),
    A2 => DSelect(0),

    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
    );



    reg0 <= "0000000000000000" when (reset ='1') else reg0_q;
    reg1 <= "0000000000000000" when (reset ='1') else reg1_q;
    reg2 <= "0000000000000000" when (reset ='1') else reg2_q;
    reg3 <= "0000000000000000" when (reset ='1') else reg3_q;
    reg4 <= "0000000000000000" when (reset ='1') else reg4_q;
    reg5 <= "0000000000000000" when (reset ='1') else reg5_q;
    reg6 <= "0000000000000000" when (reset ='1') else reg6_q;
    reg7 <= "0000000000000000" when (reset ='1') else reg7_q;

end Behavioral;
```

Screenshot of All components together:

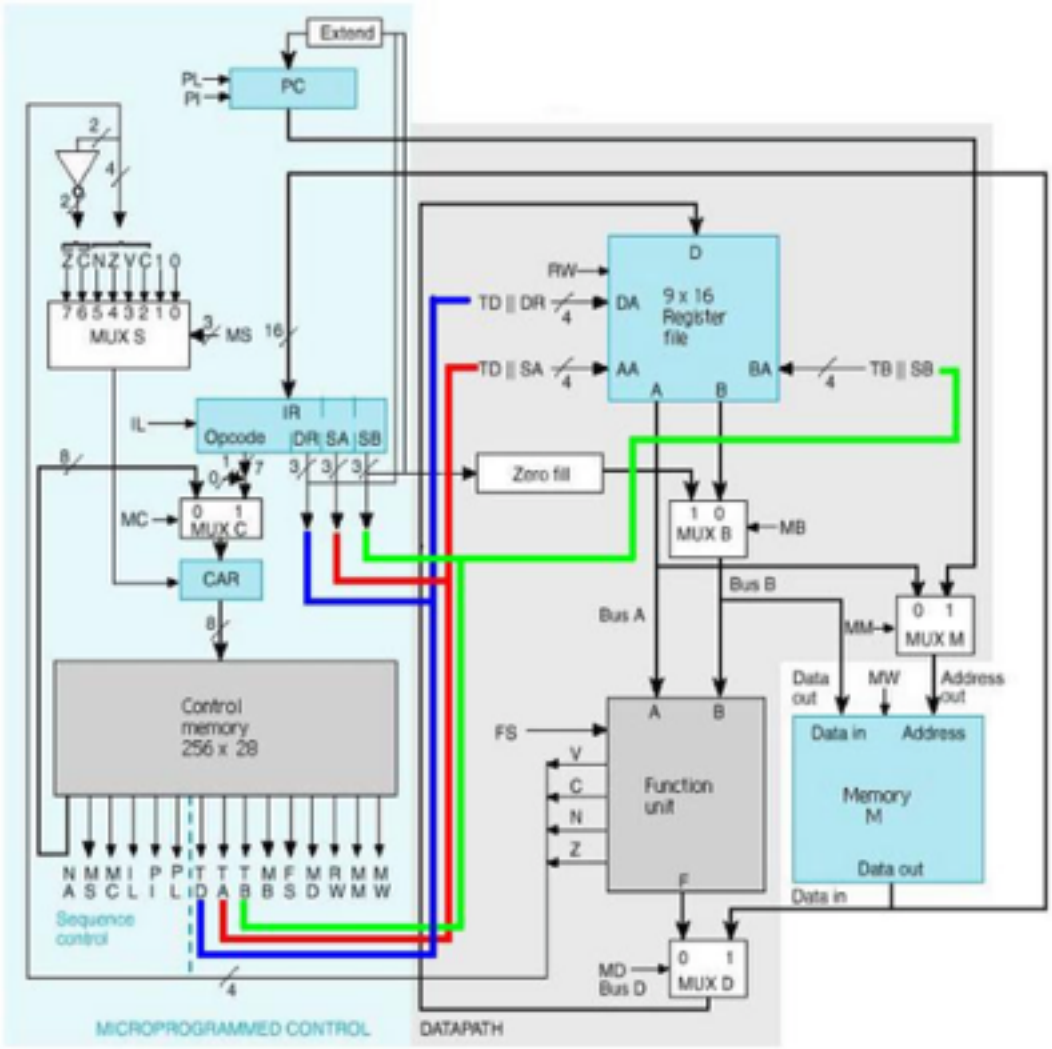


Figure 1: Multiple-Cycle Microprogrammed Control