Supplemental work for the Project:

Some of the work highlighted has been performed to show you the type of work required for you to complete. Please go through the details to comprehend the functions of each task and overall follow the instructions on the project manual and fill it out accordingly.

Sample of code:

ProgramCounter code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--Increments the program counter by 1 if there is a positive edge
clock and increment =1

entity ProgramCounter is
port (
      output : out std_logic_vector(7 downto 0);
      clk : in std_logic;
      increment : in std_logic );
end;

architecture behavior of ProgramCounter is
begin
process(clk,increment)
      variable counter: integer:=0;
begin
      if (clk'event and clk = '1' and increment = '1') then
            counter := counter + 1;
            output <= conv_std_logic_vector(counter,8);
      end if;
end process;
end behavior;
```

ControlUnit code

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

entity ControlUnit is
port (
      OpCode: in std_logic_vector(2 downto 0);
      clk: in std_logic;
      ToALoad : out std_logic;
      ToMarLoad : out std_logic;
      ToIrLoad : out std_logic;
      ToMdriLoad : out std_logic;
      ToMdroLoad : out std_logic;
      ToPcIncrement : out std_logic := '0';
      ToMarMux : out std_logic;
      ToRamWriteEnable : out std_logic;
      ToAluOp: out std_logic_vector (2 downto 0)
);
end;

architecture behavior of ControlUnit is

type cu_state_type is (load_mar, read_mem, load_mdri, load_ir, decode,
ldaa_load_mar, ldaa_read_mem, ldaa_load_mdri, ldaa_load_a, adaa_load_mar,
adaa_read_mem, adaa_load_mdri, adaa_store_load_a, staa_load_mdro,
staa_write_mem, increment_pc);

signal current_state: cu_state_type;
begin
process(clk)
begin
if (clk'event and clk='1') then
      case current_state is
            --Decode instruction
            when increment_pc =>
                  current_state <= load_mar;

            when load_mar =>
                  current_state <= read_mem;
            when read_mem =>
                  current_state <= load_mdri;
            when load_mdri =>
                  current_state <= load_ir;
            when load_ir =>
                  current_state <= decode;
```

```
            --Determines what instruction is
            when decode =>
                  if OpCode = "000" then
                        current_state <= ldaa_load_mar;
                  elsif OpCode = "001" then
                        current_state <= adaa_load_mar;
                  elsif OpCode = "010" then
                        current_state <= staa_load_mdro;
                  else
                        current_state <= increment_pc;
                  end if;

            --Load instruction
            when ldaa_load_mar =>
                  current_state <= ldaa_read_mem;
            when ldaa_read_mem =>
                  current_state <= ldaa_load_mdri;

            when ldaa_load_mdri =>
                  current_state <= ldaa_load_a;
            when ldaa_load_a =>
                  current_state <= increment_pc;

            --Add instruction
            when adaa_load_mar =>
                  current_state <= adaa_read_mem;
            --INSERT CODE HERE

            --Store instruction
            when staa_load_mdro =>
            --INSERT CODE HERE
            end case;
      end if;
end process;

process(current_state)
begin
      ToALoad <= '0';
      ToMdroLoad <= '0';
      ToAluOp <= "000";
      case current_state is
            when increment_pc =>
                  ToALoad <= '0';
                  ToPcIncrement <= '1';
                  ToMarMux <= '0';
                  ToMarLoad <= '0';
                  ToRamWriteEnable <= '0';
                  ToMdriLoad <= '0';
                  ToIrLoad <= '0';
```

```
            ToMdroLoad <= '0';
            ToAluOp <= "000";
        when load_mar =>
            ToALoad <= '0';
            ToPcIncrement <= '0';
            ToMarMux <= '0';
            ToMarLoad <= '1';
            ToRamWriteEnable <= '0';
            ToMdriLoad <= '0';
            ToIrLoad <= '0';
            ToMdroLoad <= '0';
        when read_mem =>
            ToALoad <= '0';
            ToPcIncrement <= '0';
            ToMarMux <= '0';
            ToMarLoad <= '0';
            ToRamWriteEnable <= '0';
            ToMdriLoad <= '0';
            ToIrLoad <= '0';
            ToMdroLoad <= '0';
        when load_mdri =>
            --INSERT CODE HERE

        when load_ir =>
            --INSERT CODE HERE

        when decode =>
            --INSERT CODE HERE

        when ldaa_load_mar =>
            ToALoad <= '0';
            ToPcIncrement <= '0';
            ToMarMux <= '1';
            ToMarLoad <= '1';
            ToRamWriteEnable <= '0';
            ToMdriLoad <= '0';
            ToIrLoad <= '0';
            ToMdroLoad <= '0';
            ToAluOp <= "101";
        when ldaa_read_mem =>
            --INSERT CODE HERE
        when ldaa_load_mdri =>
            ToALoad <= '0';
            ToPcIncrement <= '0';
            ToMarMux <= '0';
            ToMarLoad <= '0';
            ToRamWriteEnable <= '0';
            ToMdriLoad <= '1';
            ToIrLoad <= '0';
```

```
                ToMdroLoad <= '0';
                ToAluOp <= "101";
            when ldaa_load_a =>
                  --INSERT CODE HERE
            when adaa_load_mar =>
                ToALoad <= '0';
                ToPcIncrement <= '0';
                ToMarMux <= '1';
                ToMarLoad <= '1';
                ToRamWriteEnable <= '0';
                ToMdriLoad <= '0';
                ToIrLoad <= '0';
                ToMdroLoad <= '0';
                ToAluOp <= "000";
            when adaa_read_mem =>
                  --INSERT CODE HERE
            when adaa_load_mdri =>
                  --INSERT CODE HERE
            when adaa_store_load_a =>
                  ToALoad <= '1';
                  ToPcIncrement <= '0';
                  ToMarMux <= '0';
                  ToMarLoad <= '0';
                  ToRamWriteEnable <= '0';
                  ToMdriLoad <= '0';
                  ToIrLoad <= '0';
                  ToMdroLoad <= '0';
                  ToAluOp <= "000";
            when staa_load_mdro =>
                  ToALoad <= '0';
                  ToPcIncrement <= '0';
                  ToMarMux <= '1';
                  ToMarLoad <= '1';
                  ToRamWriteEnable <= '0';
                  ToMdriLoad <= '0';
                  ToIrLoad <= '0';
                  ToMdroLoad <= '1';
                  ToAluOp <= "100";
            when staa_write_mem =>
                  --INSERT CODE HERE
      end case;
end process;
end behavior;
```

cpu code

```
library ieee;
use ieee.std_logic_1164.all;
entity cpu is
port(
      clk: in std_logic;
      pcOut: out std_logic_vector(7 downto 0);
      marOut: out std_logic_vector (7 downto 0);
      irOutput: out std_logic_vector (7 downto 0);
      mdriOutput: out std_logic_vector (7 downto 0);
      mdroOutput: out std_logic_vector (7 downto 0);
      aOut: out std_logic_vector (7 downto 0);
      incrementOut: out std_logic );
end;

architecture behavior of cpu is
component memory_8_by_32 --memory component
port(
      clk: in std_logic;
      Write_Enable: in std_logic;
      Read_Addr: in std_logic_vector(4 downto 0);
      Data_in: in std_logic_vector(7 downto 0);
      Data_out: out std_logic_vector(7 downto 0)
);
end component;

component alu --arithmetic logic unit
port(
      A: in std_logic_vector(7 downto 0);
      B: in std_logic_vector(7 downto 0);
      AluOp: in std_logic_vector(2 downto 0);
      output: out std_logic_vector(7 downto 0)
);
end component;

component reg --register
port(
      input : in std_logic_vector(7 downto 0);
      output : out std_logic_vector(7 downto 0);
      clk : in std_logic;
      load : in std_logic
);
end component;

component ProgramCounter --program counter
port(
      increment: in std_logic;
      clk: in std_logic;
```

```
      output: out std_logic_vector(7 downto 0)
);
end component;

component TwoToOneMux --mux
port(
      A: in std_logic_vector (7 downto 0);
      B: in std_logic_vector (7 downto 0);
      address: in std_logic;
      output: out std_logic_vector (7 downto 0)
);
end component;

component sevenseg --seven segment decoder
port(
      i: in std_logic_vector(3 downto 0);
      o: out std_logic_vector(7 downto 0)
);
end component;

component ControlUnit
port(
      OpCode : in std_logic_vector(2 downto 0);
      clk : in std_logic;
      ToALoad : out std_logic;
      ToMarLoad : out std_logic;
      ToIrLoad : out std_logic;
      ToMdriLoad : out std_logic;
      ToMdroLoad : out std_logic;
      ToPcIncrement : out std_logic;
      ToMarMux : out std_logic;
      ToRamWriteEnable : out std_logic;
      ToAluOp : out std_logic_vector(2 downto 0)
);
end component;

-- Connections
signal ramDataOutToMdri : std_logic_vector(7 downto 0);
-- MAR Multiplexer connections
signal pcToMarMux : std_logic_vector(7 downto 0);
signal muxToMar : std_logic_vector  (7 downto 0);
-- RAM connections
signal marToRamReadAddr : std_logic_vector(4 downto 0);
signal mdroToRamDataIn : std_logic_vector(7 downto 0);
-- MDRI connections
signal mdriOut : std_logic_vector(7 downto 0);
-- IR connection
signal irOut : std_logic_vector(7 downto 0);
-- ALU / Accumulator connections
```

```
signal aluOut: std_logic_vector(7 downto 0);
signal aToAluB : std_logic_vector(7 downto 0);
-- Control Unit connections
signal cuToALoad : std_logic;
signal cuToMarLoad : std_logic;
signal cuToIrLoad : std_logic;
signal cuToMdriLoad : std_logic;
signal cuToMdroLoad : std_logic;
signal cuToPcIncrement : std_logic;
signal cuToMarMux : std_logic;
signal cuToRamWriteEnable : std_logic;
signal cuToAluOp : std_logic_vector(2 downto 0);

begin
Map_Memory: memory_8_by_32 port map(clk=>clk,
      Read_Addr=>marToRamReadAddr,
      Data_in=>mdroToRamDataIn,
      Data_Out=>ramDataOutToMdri,
      Write_Enable=>cuToRamWriteEnable );




-- Accumulator
--INSERT CODE HERE
-- ALU
--INSERT CODE HERE
-- Program Counter
--INSERT CODE HERE
-- Instruction Register
--INSERT CODE HERE
-- MAR mux
--INSERT CODE HERE

-- Memory Access Register
--INSERT CODE HERE
-- Memory Data Register Input
Map_MDRI: reg port map(clk=>clk,
      input=>ramDataOutToMdri,
      output=>mdriOut,
      load=>cuToMdriLoad );

-- Memory Data Register Output
--INSERT CODE HERE
-- Control Unit
--INSERT CODE HERE


--REMAINING CODE GOES HERE
```

```
pcOut <= pcToMarMux;
irOutput <= irOut;
aOut <= aToAluB;
marOut <= irOut(7 downto 5)&marToRamReadAddr;
mdriOutput <= mdriOut;
mdroOutput <= mdroToRamDataIn;
end behavior;
```

Finite State Machine Diagram: