CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

Lab Exercise: A Simple Processor

Objective:

You will create a simple processor unit that can perform basic assembly instructions in each clock cycle.
Your circuit should be able to perform the following operations:
Loading specific data into a register, or move immediate instruction (`mvi`)
Moving data from one register to another (`mv`)
Addition of data in registers (`add`)
Subtraction of data in registers (`sub`)

PART 1

Your will build a processor that consists of the following elements (shown in figure 1):

- Registers R0 to R7
- A multiplexer
- An adder/subtractor unit
- Register A
- Register G
- IR register
- A control unit

It will have the following inputs:

- Clock, Run, Resetn, DIN

And the following outputs:

- Bus, Done

Functionality:

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM

Instructor: Professor Izidor Gertner

- Data is loaded from DIN to the 16-bit wide multiplexer into the various registers (R0, …, R7, and A). The multiplexer also lets you transfer data from one register onto another. Notice how the output wires of the multiplexer are called a bus in the figure. This is because this term is often used for wiring that allows data to be transferred from one location in a system to another.

- To add/subtract you would use the multiplexer to first place one 16-bit number onto the bus wires and loading this number into register A. Then you would place another 16-bit number onto the bus. After this you would perform the operation specified and the result should be loaded into register G. The data in G can then be transferred to one of the other registers as specified by your instruction. Addition and subtraction take more than one clock cycle to complete because multiple transfers should be performed across the bus.

- The control unit determines when certain data has to be placed onto the bus wires and which of the registers is to be loaded with this data. For example, if the control unit asserts the signals $R0_{out}$ and A_{in}, then the multiplexer will place the contents of register R0 onto the bus and this data will be loaded by the next active clock edge into register A.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

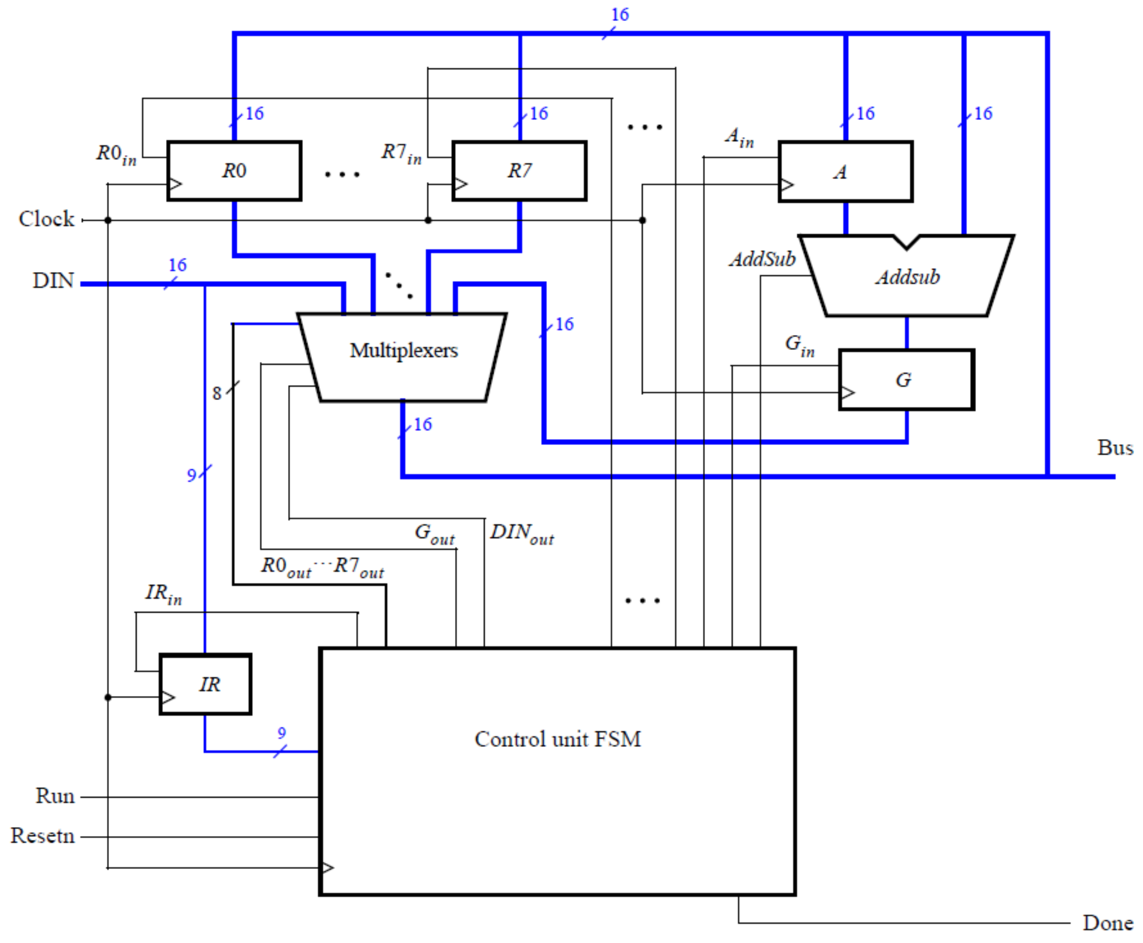


Figure 1. Simple Processor Schematic View

- The instructions that your simple processor has to support for part 1 of this lab are listed in Figure 2. The meaning of the syntax RX ← [RY] is that the contents of register RY are loaded into register RX. The **`mv`** (move) instruction allows data to be copied from one register to another. For the **`mvi`** (move immediate) instruction the expression RX ← D indicates that the 16-bit constant D is loaded into register RX.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

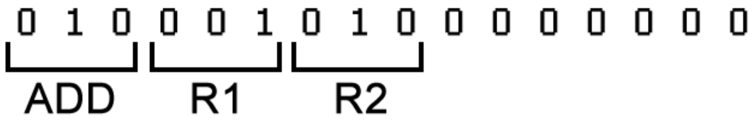
Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi $Rx,\#D$	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Figure 2. Instructions performed by the simple processor

- The instructions are encoded and stored in the IR register using the following 9-bit format: IIIXXXYYY, where III represents the instruction, XXX the RX register, and YYY the RY register. For example, if we input the following 16-bit data into DIN:

  0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0

  , the leftmost bits of DIN are connected to IR, so this sequence would translate as:



A suggested binary-coded list of operations is shown below:

Instruction	Binary equivalence
mv	000
mvi	001
add	010
sub	011

Figure 3. Suggested binary equivalence of instructions

Notice that we are using 3 bits for the instructions. Even though we are implementing only 4 instructions which can be represented using just 2 bits, this approach allows us to add more instructions later on.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

- Some instructions, such as **<u>addition or subtraction, take more than one clock cycle to complete</u>**, because multiple transfers have to be performed across the bus. The control unit can be seen as a finite state machine that "steps through" such instructions, asserting the control signals needed in successive clock cycles until the instruction has completed. The processor starts executing the instruction on the DIN input when the Run signal is asserted and the processor asserts the *Done* output when the instruction is finished. Figure 4 indicates the control signals that can be asserted in each time step to implement the instructions listed in Figure 2. Note that the only control signal asserted in time step 0 is IR_{in}, so this time step is not shown in figure 4.

	T_1	T_2	T_3
(mv): I_0	RY_{out}, RX_{in}, *Done*		
(mvi): I_1	DIN_{out}, RX_{in}, *Done*		
(add): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	G_{out}, RX_{in}, *Done*
(sub): I_3	RX_{out}, A_{in}	RY_{out}, G_{in}, *AddSub*	G_{out}, RX_{in}, *Done*

Figure 4. Control signals asserted in each instruction/time step

- As we said before, the control unit can be seen as a finite state machine. It can be described with the state diagram in Figure 5. For instance, the diagram shows that the system starts at the (T0) state and get the opcode for the instruction from input, then based on the opcode, the system will move to either the Move(T1) or Add(T2) state. If the system is at the Add (T2) state, it will automatically move to the T3 state on the next clock cycle. Then, the system will return to the Decode state when the *Done* signal is set to 1 and the instruction register has been enable. Thereby, the above diagram shows that both the **mv** and the **mvi** commands can be seen as if combined into one state, Move. Similarly, both the **add** and **sub** instructions can be regarded as combined into two states, Add1 and Add2.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

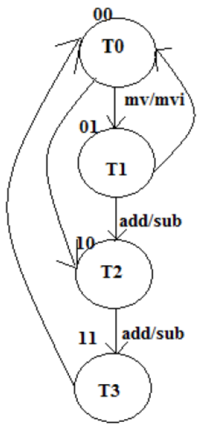


Figure 5. Control Unit FSM

Your Task:

1. Create a new Quartus II project.

2. Design your circuit with the functionality described above using Figure 1 as reference. To help you get started, we provide you with some skeleton VHDL code in Appendix A of this lab. If you would like to work using block diagram design instead, feel free to do so. We have also included a snapshot of our top cpu circuit in Block diagram design in the Appendix.

3. Run a functional simulation to verify that the behavior of your code is correct. A sample simulation is shown in figure 4. DIN is shown in hexadecimal representation. IR is shown in 3-digit octal numbers. Some interpretations:

   - Notice how at 30ns the Hex value 2000 is loaded into IR from DIN.
     The pattern 2000_{16} in binary is equivalent to 0010 0000 0000 0000. Taking the 9 leftmost digits for IR, we read 001 000 000, which according to our binary equivalences in Figure 6, it represents the instruction mvi R0, #D, where the value D=5 is loaded into R0 on the rising edge of the clock at 50ns.

   - At 90ns the simulation shows the instruction 0400, which is equivalent to mv R1,R0. We see how the value 0005 in register R0 was transferred to R1. The output *Done* notifies if and when the instruction operation was executed successfully.

- Similarly, at 110ns we see the operation add R0,R1. And at 190ns, the operation sub R0,R0. Notice as well that both add and sub operations take longer cycles to complete. This is exemplified accordingly by the *Done* output wave.

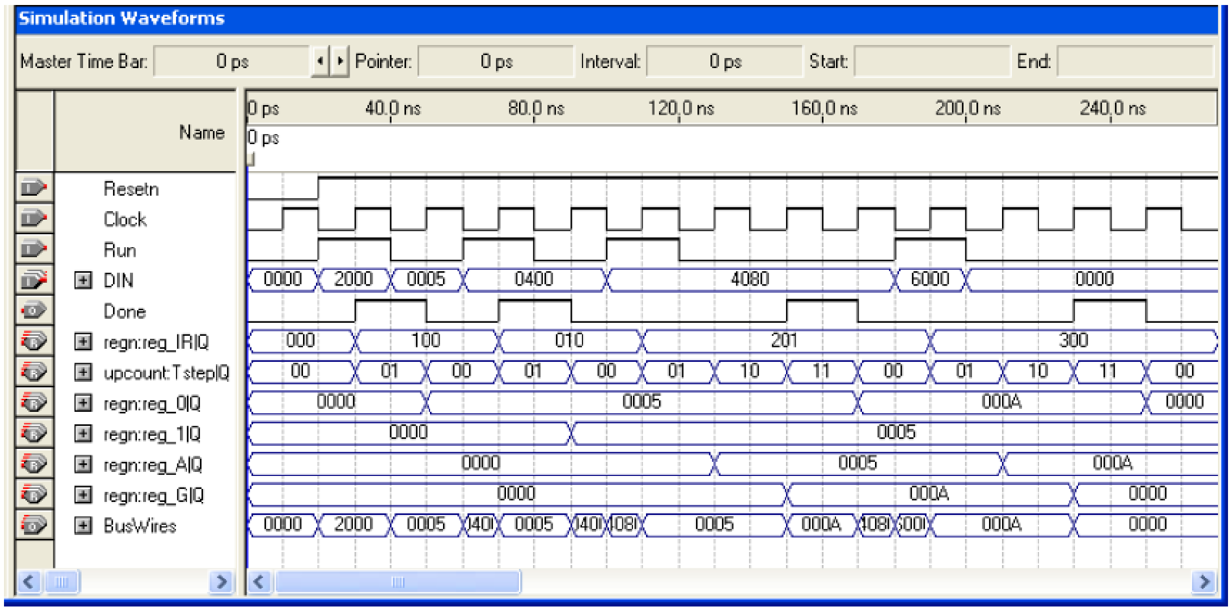


Figure 6. Sample simulation for processor circuit

4. Create a new Quartus file which will be used as a top entity that will implement the processor that you just built into the DE2 board. This file should contain the appropriate input and output port connections. An skeleton code for this top level is provided in the Appendix of this lab.

5. Pin Assignment: Click here for the pin assignment manual

   Create a new text file to add your pin assignments.

   - 16-bit wide DIN input: Assign to SW15-0.
   - Run: Assign to SW17.
   - Clock: Assign to KEY1.
   - Resetn: Assign to KEY0.
   - Bus wires: Connect to LEDR15-0.
   - Done signal: Connect to LEDR17.

Note: Instead of connecting your outputs to be displayed in the LEDs, you may want to output to the 7-segment display instead. Feel free to do so. The complete VHDL code for a 4-to-7 decoder is given in the Appendix if you wish to do it this way.

6. Compile your circuit.
   You should be ready to test the functionality on the DE2 board. Try performing different operations with diverse registers to test your circuit. For example, you may want to try the following operations:

Load a specific value onto R0:	mvi R0, #D ; where D = any number you please
Move the value from R0 to R1:	mv R1, R0
Load a specific value onto R2:	mvi R2, #D ; where D = any number you please
Add R1+R2 onto R2:	add R2, R1
Subtract R1-R1:	sub R1,R1

PART 2

In this part you are to design the circuit depicted in Figure 7, in which a memory module and counter are connected to the processor from Part I. The counter is used to read the contents of successive addresses in the memory, and this data is provided to the processor as a stream of instructions. To simplify the design and testing of this circuit we have used separate clock signals, PClock and MClock, for the processor and memory.

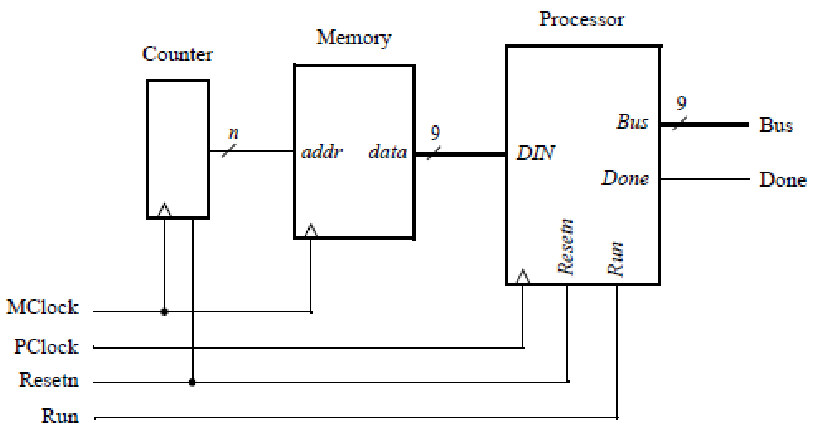


Figure 7. The processor connected to a memory and a counter

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

<u>Your Task:</u>

1. Create a new Quartus Project.

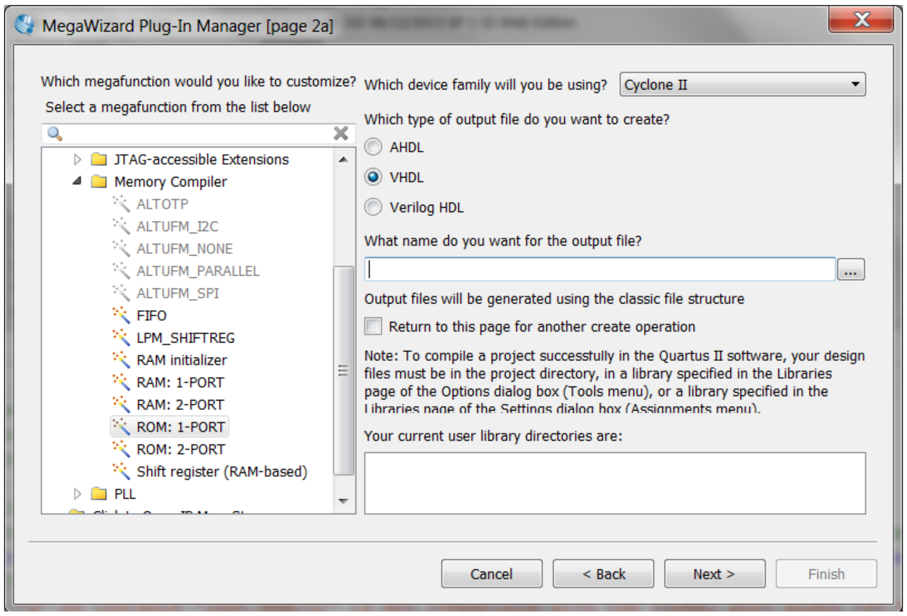
2. Create a MIF file. You can just create it in Notepad, change the extension to *.mif and add this text file to your working directory. A sample mif file is given in the appendix.
   You must make the necessary changes in the MIF file given to test your own operations with various registers.

   Why do we need a MIF file? To place processor instructions into the memory, you need to specify initial values that should be stored in the memory once your circuit has been programmed into the FPGA chip. This can be done by telling the wizard to initialize the memory using the contents of a memory initialization file (MIF).

3. Create the memory module using the Quartus Mega-Wizard Plug-In Manager Tool to create the memory module from the library of parameterized modules (LPMs). To do so, you can follow these steps:

   - Go to Tools > Mega-Wizard Plugin Manager > Next
   - You can find the ROM module under Memory Compiler > ROM 1 –PORT, give a name to your file.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

- JTAG-accessible Extensions
- Memory Compiler
  - ALTOTP
  - ALTUFM_I2C
  - ALTUFM_NONE
  - ALTUFM_PARALLEL
  - ALTUFM_SPI
  - FIFO
  - LPM_SHIFTREG
  - RAM initializer
  - RAM: 1-PORT
  - RAM: 2-PORT
  - ROM: 1-PORT
  - ROM: 2-PORT
  - Shift register (RAM-based)
- PLL

Which device family will you be using? Cyclone II

Which type of output file do you want to create?
AHDL [✓ VHDL] Verilog HDL

What name do you want for the output file?

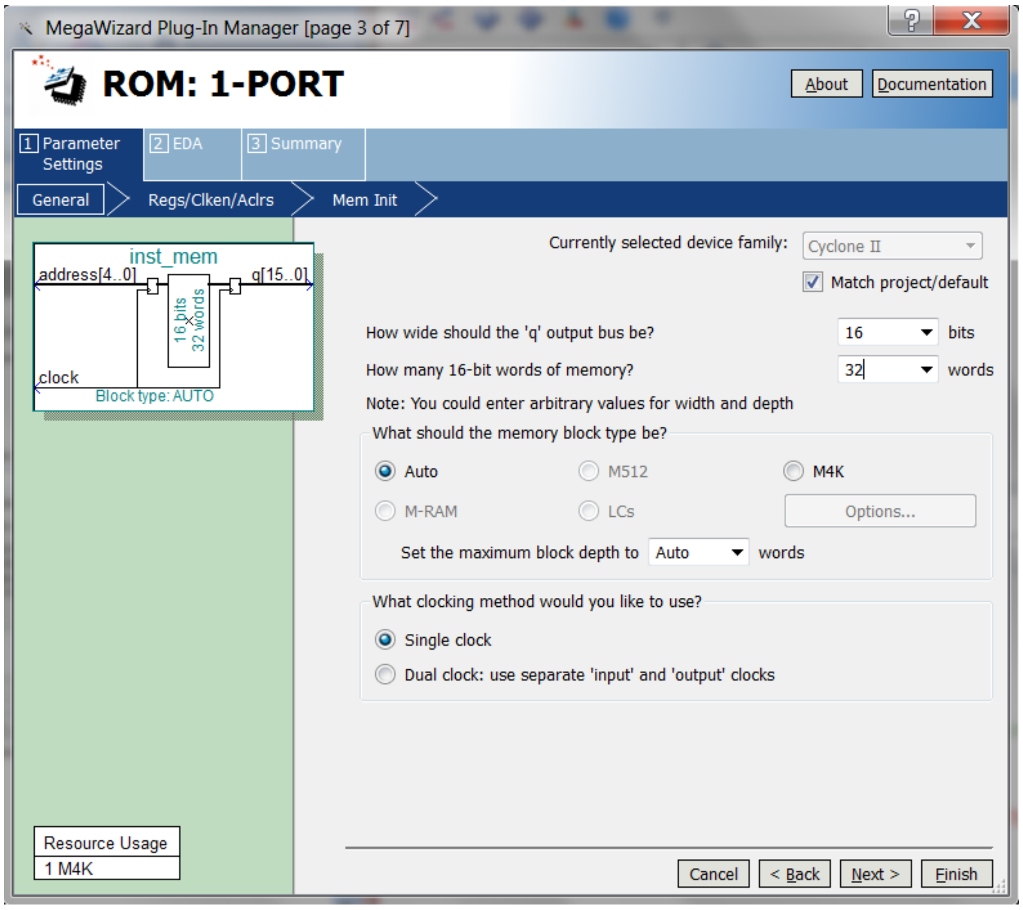
Output files will be generated using the classic file structure

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

- Follow the values of the picture below. The 'q' outputs should be 16 bits, and 32 words of memory.



Currently selected device family: Cyclone II

[✓] Match project/default

How wide should the 'q' output bus be? 16 bits

How many 16-bit words of memory? 32 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

[✓ Auto] M512 M4K M-RAM LCs Options...

Set the maximum block depth to Auto words

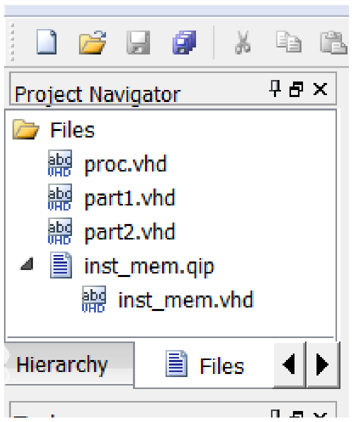
What clocking method would you like to use?

[✓ Single clock]

Dual clock: use separate 'input' and 'output' clocks

- Click Next, then **uncheck 'q' output port**. Click next again.
- Browse for your mif file, then click Next. Click Next again, then Finish. By default, Altera will ask you if you want to include the file that has just been created into your working project, click YES. You should now see a qip reference file in your navigation panel in Quartus.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner



4. Create a top-level file that instantiates the processor, memory and counter. A skeleton for this file is provided in the appendix. See part2.vhd
5. Use functional simulation to test the circuit. Ensure that data is read properly out of the ROM and executed by the processor.
6. Pin Assignment: Click here for the pin assignment manual

   Create a new text file to add your pin assignments.
   - Run: Assign to SW17.
   - Resetn: Assign to KEY0.
   - MClock: Assign to KEY1.
   - PClock: Assign to KEY2.
   - Bus wires: Connect to LEDR15-0.
   - Done signal: Connect to LEDR17.
7. Compile your circuit and use functional simulation to test it. Check that the data is read properly out of the ROM and executed by the processor.

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

APPENDIX

proc.vhd

```
-- proc.vhd (Processor file - will be connected to top-entity PART1)
LIBRARY ieee; USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY proc IS
      generic (
            mv : std_logic_vector(2 downto 0) := "000";
            mvi : std_logic_vector(2 downto 0) := "001";
            add : std_logic_vector(2 downto 0) := "010";
            sub : std_logic_vector(2 downto 0) := "011"
      );
      PORT (
             --TO DO: Declare the following ports:
             --DIN
             --Resetn, Clock, Run

             Done : BUFFER STD_LOGIC;
             BusWires : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0)
      );
END proc;

ARCHITECTURE Behavior OF proc IS
      COMPONENT dec3to8
            PORT (W : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                  En : IN STD_LOGIC;
                  Y : OUT STD_LOGIC_VECTOR(0 TO 7));
      END COMPONENT;


      COMPONENT regn
            GENERIC (n : INTEGER := 16);
            PORT ( R : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
                   Rin, Clock : IN STD_LOGIC;
                   Q : BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0));
      END COMPONENT;

      TYPE State_type IS (T0, T1, T2, T3);
      SIGNAL Sel : STD_LOGIC_VECTOR(1 to 10); -- bus selector
      SIGNAL Tstep_Q, Tstep_D: State_type;
      SIGNAL I : STD_LOGIC_VECTOR(2 DOWNTO 0);
      --TO DO: Complete signal declarations:
      --R0, R1, R2, R3, R4, R5, R6, R7, A, G
      --IR
      --Rin, Rout
      --Sum
      --High, IRin, DINout, Ain, Gin, Gout, AddSub
      --Xreg, Yreg

BEGIN
      High <= '1';
      I <= IR(1 TO 3);
      decX: dec3to8 PORT MAP (IR(4 TO 6), High, Xreg);
      decY: dec3to8 PORT MAP (IR(7 TO 9), High, Yreg);

      statetable: PROCESS(Tstep_Q, Run, Done)
      BEGIN
            CASE Tstep_Q IS
                  WHEN T0 =>  -- data is loaded into IR in this time step
                              IF(Run = '0') THEN Tstep_D <=T0;
                              ELSE Tstep_D <= T1;
                              END IF;
                  WHEN T1 =>  -- some instructions end after this time step
                              -- TO DO: IF (Done = '1') THEN ...
                  WHEN T2 =>  -- always go to T3 after this
                              Tstep_D <= T3;
                  WHEN T3 =>  -- instructions end after this time step
                              Tstep_D <= T0;
            END CASE;
      END PROCESS;

      controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
      BEGIN
            -- TO DO: Specify initial values for
            -- Done, Ain, Gin, Gout, AddSub, IRin, Rin, Rout, DINout
            Done <= '0'; --etc...

            CASE Tstep_Q IS
                  WHEN T0 => -- store DIN in IR as long as Tstep_Q = 0
                        IRin <= '1';
                  WHEN T1 => -- define signals in time step T1
                        CASE I IS
                              WHEN mv => -- mv Rx,Ry
                                    Rout <= Yreg;
                                    Rin <= Xreg;
                                    Done <= '1';
                              WHEN mvi => -- mvi Rx,#D
                                    -- data is required to be on DIN
                                    --TO DO: complete this
                              WHEN add => -- add
                                    --TO DO: complete this
                              WHEN OTHERS => -- sub
                                    Rout <= Xreg;
                        Ain <= '1';
                END CASE;
        WHEN T2 => -- define signals in time step T2
                CASE I IS
                        WHEN add => -- add
                                Rout <= --TO DO: complete this
                                Gin <= --TO DO: complete this
                        WHEN OTHERS => -- sub
                                Rout <= --TO DO: complete this
                                AddSub <= --TO DO: complete this
                                Gin <= --TO DO: complete this
                END CASE;
        WHEN T3 => -- define signals in time step T3
                CASE I IS
                        WHEN add => -- add
                                --TO DO: complete this
                        WHEN OTHERS => -- sub
                                --TO DO: complete this
                END CASE;
        END CASE;
END PROCESS;

fsmflipflops: PROCESS (Clock, Resetn, Tstep_D)
BEGIN
        IF (Resetn = '0') THEN
                Tstep_Q <= T0;
        ELSIF (rising_edge(Clock)) THEN
                Tstep_Q <= Tstep_D;
        END IF;
END PROCESS;

reg_0: regn PORT MAP (BusWires, Rin(0), Clock, R0);
--TO DO: ...instantiate other registers (1-7,A,IR)


--alu
alu: PROCESS (AddSub, A, BusWires)
BEGIN
        IF AddSub = '0' THEN
                Sum <= --TO DO: complete this ;
        ELSE
                --TO DO: complete this ;
        END IF;
END PROCESS;

reg_G: regn PORT MAP (Sum, Gin, Clock, G);


-- define the internal processor bus
Sel <= Rout & Gout & DINout;
```

```
      busmux: PROCESS (Sel, R0, R1, R2, R3, R4, R5, R6, R7, G, DIN)
      BEGIN
            IF Sel = "1000000000" THEN BusWires <= R0;
            --TO DO: complete the bus definition
      END PROCESS;
END Behavior;



LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec3to8 IS
      PORT ( W : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
             En: IN STD_LOGIC;
             Y : OUT STD_LOGIC_VECTOR(0 TO 7));
END dec3to8;

ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
      PROCESS (W, En)
      BEGIN
            IF En = '1' THEN
                  CASE W IS
                        WHEN "000" => Y <= "10000000";
                        WHEN "001" => Y <= "01000000";
                        WHEN "010" => Y <= "00100000";
                        WHEN "011" => Y <= "00010000";
                        WHEN "100" => Y <= "00001000";
                        WHEN "101" => Y <= "00000100";
                        WHEN "110" => Y <= "00000010";
                        WHEN "111" => Y <= "00000001";
                  END CASE;
            ELSE
                  Y <= "00000000";
            END IF;
      END PROCESS;
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
      GENERIC (n : INTEGER := 16);
      PORT ( R : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
             Rin, Clock : IN STD_LOGIC;
             Q : BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END regn;
```

```
ARCHITECTURE Behavior OF regn IS
BEGIN
      PROCESS (Clock)
      BEGIN
            IF Clock'EVENT AND Clock = '1' THEN
                  IF Rin = '1' THEN
                        Q <= R;
                  END IF;
            END IF;
      END PROCESS;
END Behavior;
```

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

part1.vhd

```
-- part1.vhd (TOP ENTITY WITH PORT CONNECTIONS)
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part1 IS
PORT ( KEY  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
       --TO DO: Enter the port vector description for:
       --SW
       --LEDR
      );
END part1;

ARCHITECTURE Behavior OF part1 IS
      COMPONENT proc
            PORT (--TO DO: Enter the port vector description for:
                  --DIN ;
                  --Resetn, Clock, Run ;
                  Done  : BUFFER STD_LOGIC;
                  BusWires : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0));
      END COMPONENT;
      SIGNAL Manual_Clock, Resetn, Run, Done : STD_LOGIC;
      SIGNAL DIN, BusWires : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
      Resetn <= --TO DO: Enter the KEY assignment for Resetn ;
      Manual_Clock <= --TO DO: Enter the KEY assignment for clock ;
            -- Note: can't use name Clock because this is defined as
            -- the 50 MHz Clock coming into the FPGA from the board
      DIN <= --TO DO: What switches are assigned to DIN? ;
      Run <= --TO DO: What switch is assigned to RUN? ;

      -- proc(DIN, Resetn, Clock, Run, Done, BusWires);
      U1: proc PORT MAP (DIN, Resetn, Manual_Clock, Run, Done, BusWires);

      --TO DO: Declare behavior of LEDR(15-0), LEDR(17)
      LEDR(16) <= '0';
END Behavior;
```

inst_mem.mif (sample mif file, develop your own values)

```
DEPTH = 32;
WIDTH = 16;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
     00   :    0010000000000000;          % mvi R0,5 %
     01   :    0000000000000101;
     02   :    0000010000000000;          % mv  R1,R0 %
     03   :    0100000010000000;          % add R1,R0 %
     04   :    0110000000000000;          % sub R0,R0 %
     05   :    0000000000000000;
     06   :    0000000000000000;
     07   :    0000000000000000;
     08   :    0000000000000000;
     09   :    0000000000000000;
     0A   :    0000000000000000;
     0B   :    0000000000000000;
     0C   :    0000000000000000;
     0D   :    0000000000000000;
     0E   :    0000000000000000;
     0F   :    0000000000000000;
     10   :    0000000000000000;
     11   :    0000000000000000;
     12   :    0000000000000000;
     13   :    0000000000000000;
     14   :    0000000000000000;
     15   :    0000000000000000;
     16   :    0000000000000000;
     17   :    0000000000000000;
     18   :    0000000000000000;
     19   :    0000000000000000;
     1A   :    0000000000000000;
     1B   :    0000000000000000;
     1C   :    0000000000000000;
     1D   :    0000000000000000;
     1E   :    0000000000000000;
     1F   :    0000000000000000;
END;
```

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70

Due by April 30, 2015, at 11:59 PM

Instructor: Professor Izidor Gertner

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner

part2.vhd

```
-- part2.vhd (top level file with port connections and counter)
-- Reset with KEY(0). Clock counter and memory with KEY(2). Clock
-- each instuction into the processor with KEY(1). SW(17) is the Run input.
-- Use KEY(2) to advance the memory as needed before each processor KEY(1)
-- clock cycle.
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part2 IS
PORT (
      --TO DO: Declare the following ports
      --KEY, SW, LEDR );
END part2;

ARCHITECTURE Behavior OF part2 IS
      COMPONENT proc
            PORT ( --TO DO: Declare ports:
                   --DIN
                   --Resetn,Clock,Run
                   --Done
                   --BusWires
                 );
      END COMPONENT;
      COMPONENT inst_mem
            PORT ( address : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                   clock : IN STD_LOGIC ;
                   q : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
      END COMPONENT;
      COMPONENT count5
            PORT ( Resetn, Clock : IN STD_LOGIC;
                   Q : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
      END COMPONENT;

      SIGNAL pc : STD_LOGIC_VECTOR(4 DOWNTO 0);
      --TO DO: Declare signals:
      --Resetn, PCloc, Mclock, Run, Done
      --DIN, BusWires

BEGIN
      --TO DO: Declare behavior of Resetn, PClock, MClock, Run

      U1: proc PORT MAP (DIN, Resetn, PClock, Run, Done, BusWires);

      --TO DO: Declare behavior of LEDR(15-0), LEDR(17)
      LEDR(16) <= '0';

      U2: inst_mem PORT MAP (pc, MClock, DIN);
      U3: count5 PORT MAP (Resetn, MClock, pc);
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY count5 IS
PORT ( Resetn, Clock : IN STD_LOGIC;
       Q : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END count5;

ARCHITECTURE Behavior OF count5 IS
      SIGNAL Count : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
      PROCESS (Clock, Resetn)
      BEGIN
                  IF (Resetn = '0') THEN
                        Count <= --TO DO: What would be the value of Count? ;
                  ELSIF (rising_edge(Clock)) THEN
                        Count <= Count + '1';
                  END IF;
      END PROCESS;
      Q <= Count;
END Behavior;
```

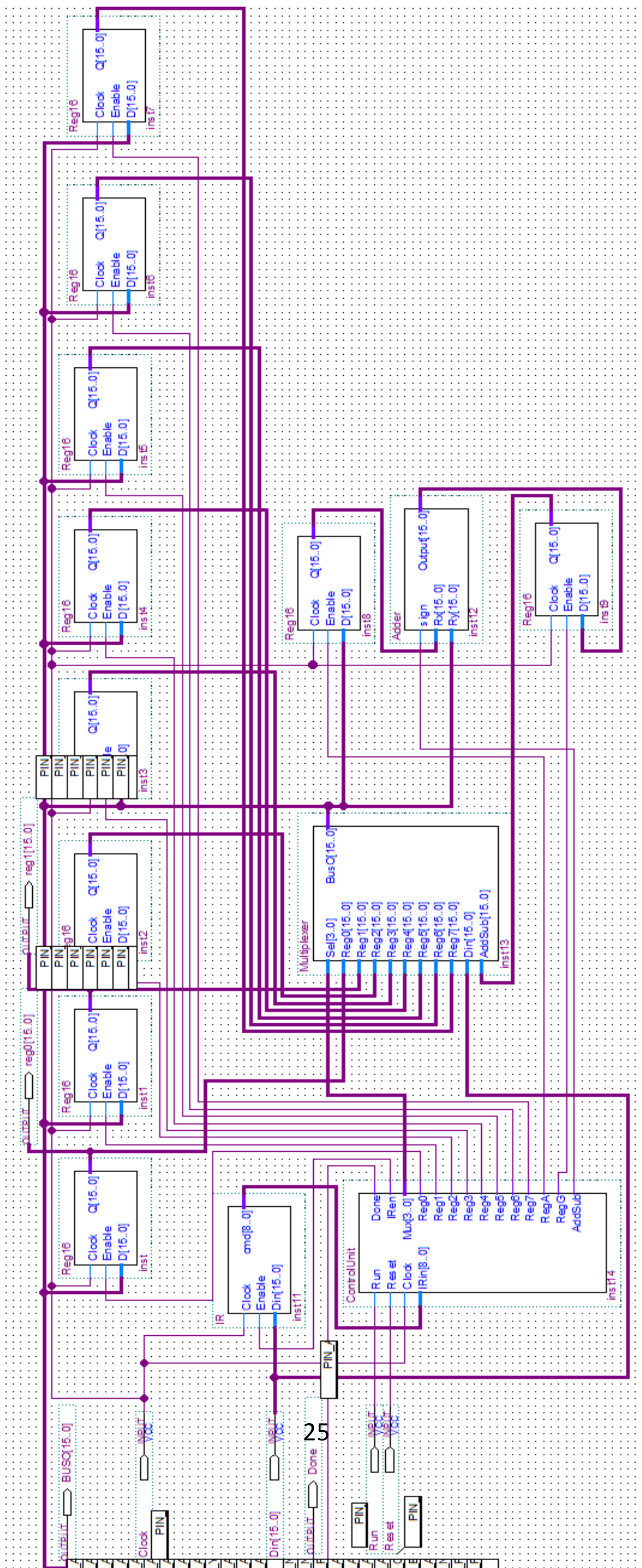
dec_7seg.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
-- Hexadecimal to 7 Segment Decoder for LED Display

ENTITY dec_7seg IS
      PORT( hex_input : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      segment_a, segment_b, segment_c, segment_d, segment_e, segment_f,
      segment_g : OUT std_logic);
END dec_7seg;
ARCHITECTURE a OF dec_7seg IS
      SIGNAL segment_data : STD_LOGIC_VECTOR(6 DOWNTO 0);
      BEGIN
            PROCESS (hex_input)
            -- HEX to 7 Segment Decoder for LED Display
            BEGIN -- Hex-digit is the four bit binary value to display in
            hexadecimal
            CASE hex_input IS
                  WHEN "0000" =>
                  segment_data <= "1111110";
                  WHEN "0001" =>
                  segment_data <= "0110000";
                  WHEN "0010"=>
                  segment_data <= "1101101";
                  WHEN "0011" =>
                  segment_data <= "1111001";
                  WHEN "0100" =>
                  segment_data <= "0110011";
                  WHEN "0101" =>
                  segment_data <= "1011011";
                  WHEN "0110" =>
                  segment_data <= "0011111";
                  WHEN "0111" =>
                  segment_data <= "1110000";
                  WHEN "1000" =>
                  segment_data <= "1111111";
                  WHEN "1001" =>
                  segment_data <= "1111011";
                  WHEN "1010" =>
                  segment_data <= "1110111";
                  WHEN "1011" =>
                  segment_data <= "0011111";
                  WHEN "1100" =>
                  segment_data <= "1001110";
                  WHEN "1101" =>
                  segment_data <= "0111101";
                  WHEN "1110" =>
                  segment_data <= "1001111";
                  WHEN "1111" =>
                  segment_data <= "1000111";

            END CASE;
            END PROCESS;
      -- extract segment data bits and invert
      -- LED driver circuit is inverted
      segment_a <= NOT segment_data(6);
      segment_b <= NOT segment_data(5);
      segment_c <= NOT segment_data(4);
      segment_d <= NOT segment_data(3);
      segment_e <= NOT segment_data(2);
      segment_f <= NOT segment_data(1);
      segment_g <= NOT segment_data(0);
END a;
```

CS 343
Simple Procedssor
in VHDL and Model-SIM , Quartus, DE 2-70
Due by April 30, 2015, at 11:59 PM
Instructor: Professor Izidor Gertner



Reg16
Clock
Enable
D[15..0]
Q[15..0]
inst
inst1
inst2
inst3
inst4
inst5
inst6
inst7
inst8
inst9
Adder
sign
Rx[15..0]
Ry[15..0]
Output[15..0]
inst12
Multiplexer
Sel[3..0]
Reg0[15..0]
Reg1[15..0]
Reg2[15..0]
Reg3[15..0]
Reg4[15..0]
Reg5[15..0]
Reg6[15..0]
Reg7[15..0]
Din[15..0]
AddSub[15..0]
BusQ[15..0]
inst13
IR
cmd[8..0]
Din[15..0]
inst11
ControlUnit
Run
Reset
Clock
IRin[8..0]
Done
IRen
Mux[3..0]
Reg0
Reg1
Reg2
Reg3
Reg4
Reg5
Reg6
Reg7
RegA
RegG
AddSub
inst14
PIN
OUTPUT
reg0[15..0]
reg1[15..0]
BUSQ[15..0]
Done
VCC
Clock
Din[15..0]
Run
Reset