COE608 Computer Organization and Architectures
Winter 2017
Lab 4b: Data-Path Design

Due Date: Week 9 (During your lab session)

1 Objectives

The objective of this lab is to build and test the data-path for a 32-bit processor. Students have designed and unit tested most of the components (PC, Registers, data memory and ALU) required for data-path in the previous labs. The block diagram of the data-path is illustrated in Figure 1. In this lab, students will interconnect these components together to create the CPU's data-path. Sufficient testing will also need to be done to ensure that the CPU data-path functions correctly. CPU specifications are described in another document, *CPU Specification* which is available on the lab webpage which outline instruction specification to help you assert the correct control signals for testing. Note that the *Instruction Memory,* M[INST] will be implemented in Lab6.

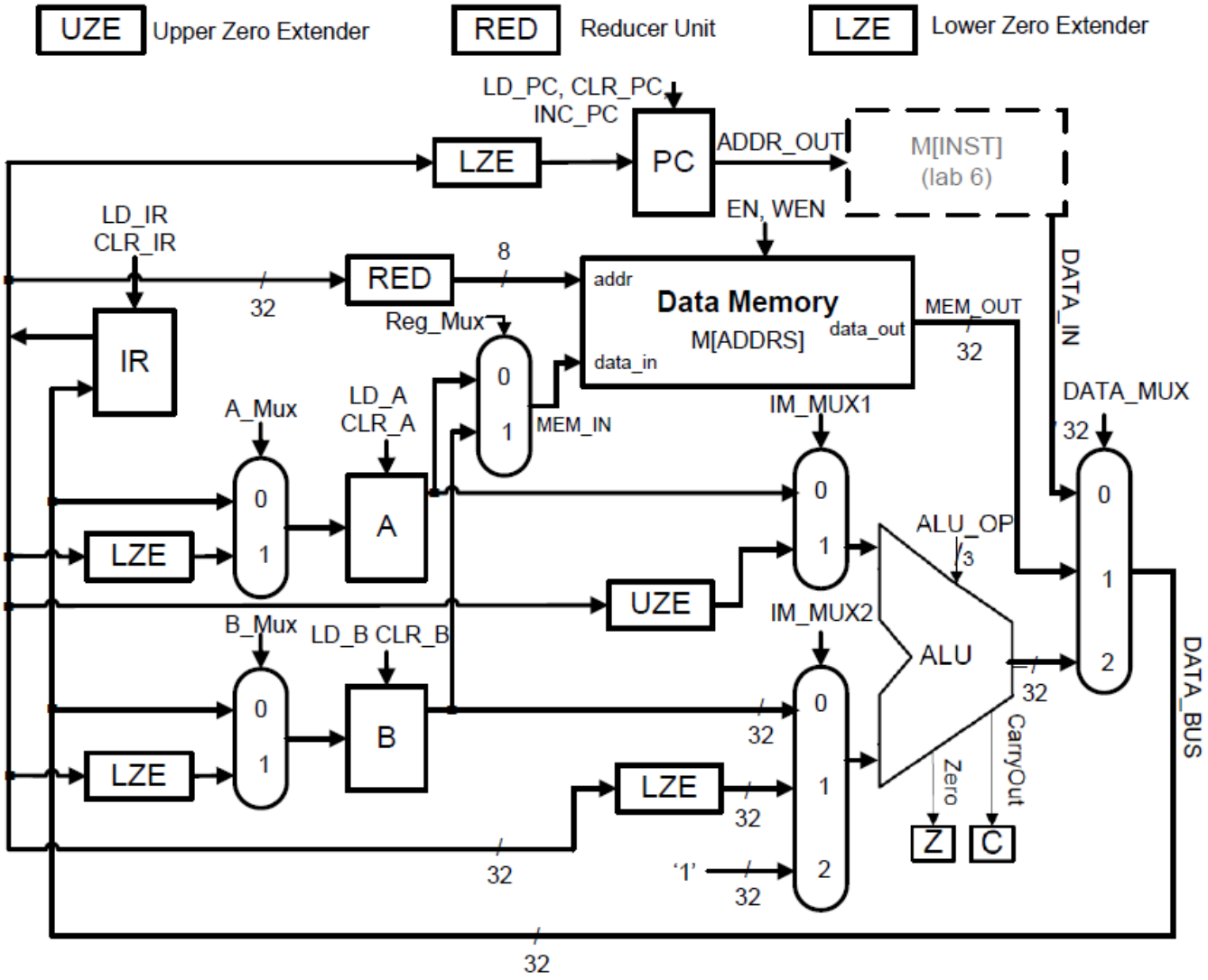

Figure 1: CPU Data-Path

2 Data-Path Implementation

Students will implement the CPU data-path of Figure 1 in VHDL. You may use the previously designed components such as the PC, Registers, ALU and Data-Memory unit by instantiation using the component and "Port Map" command. The CPU data-path declaration should look similar to what is given in the following:

```
-------------------------------------------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY datapath IS
PORT(  Clk, mClk  :          IN STD_LOGIC;  -- clock Signal

--Memory Signals
WEN, EN :                    IN STD_LOGIC;

-- Register Control Signals (CLR and LD).
Clr_A , Ld_A :               IN STD_LOGIC;
Clr_B , Ld_B :               IN STD_LOGIC;
Clr_C , Ld_C :               IN STD_LOGIC;
Clr_Z , Ld_Z :               IN STD_LOGIC;
Clr_PC , Ld_PC :             IN STD_LOGIC;
Clr_IR , Ld_IR :             IN STD_LOGIC;

-- Register outputs (Some needed to feed back to control unit. Others pulled out for testing.
Out_A :                      OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
Out_B :                      OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
Out_C :                      OUT STD_LOGIC;
Out_Z :                      OUT STD_LOGIC;
Out_PC :                     OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
Out_IR :                     OUT STD_LOGIC_VECTOR(31 DOWNTO 0);

-- Special inputs to PC.
Inc_PC :                     IN STD_LOGIC;

-- Address and Data Bus signals for debugging.
ADDR_OUT :                          OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
DATA_IN  :                          IN STD_LOGIC_VECTOR(31 DOWNTO 0);
DATA_OUT, MEM_OUT, MEM_IN :         OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
MEM_ADDR :                          OUT STD_LOGIC_VECTOR(7 DOWNTO 0);

-- Various MUX controls.
DATA_Mux:                    IN STD_LOGIC_VECTOR(1 DOWNTO 0);
REG_Mux :                    IN STD_LOGIC;
A_MUX, B_MUX :               IN STD_LOGIC;
IM_MUX1 :                    IN STD_LOGIC;
IM_MUX2 :                    IN STD_LOGIC_VECTOR(1 DOWNTO 0);
```

```
-- ALU Operations.
ALU_Op :                IN STD_LOGIC_VECTOR(2 DOWNTO 0));

END datapath;
-------------------------------------------------------------------------------------------------------------------
ARCHITECTURE description OF datapath IS
--  Component instantiations        --  you figure this out
--  Internal signals                --  you decide what you need

BEGIN
--  you fill in the details

END description;
-------------------------------------------------------------------------------------------------------------------
```

Please note that all the Register outputs are declared as output signals. This is done to allow for system testing and to ensure correct operation of the CPU, its data-path, and to observe the status of the internal registers. A full computer system would use various hardware and software utilities to display this information to the user. However, in this simplified case, this option is not available.

To avoid having to tell Quartus II about the file location paths, simply import the previously written VHDL source files by copying and pasting the .vhd files of the components used in this lab to the same working directory as the data-path project. Quartus II will look into its project directory for the required components.

3 Control Signals

The data-path requires numerous control signals. These control signals include clear and load for all the Registers and the PC, the ALU operation, and signals for several MUXes (a note on multiplexers later) to control the flow of data. The clock (Clk) is not shown on the block diagram. It is assumed that the clock is connected to all the sequential components - namely the PC and Registers. The mClk is used for the data memory module. The mClk must operate at twice the frequency of the data-path clock in order to read and write data (i.e. 2*mClk = Clk, mClk = 20ns, Clk = 40ns) within a single pipeline stage. Note that mClk's falling edge should then align with Clk's rising edge. Use the timing characteristics of lab 4a as a basis for your timing simulations in this lab. Ensure that the setup and hold requirements of your CPU components are met, and take propagation delays into consideration when designing the waveforms for testing.

Table 1 listed at the end of this document shows a listing of the operations that the processor must be capable of supporting in order to function correctly. Note that the PC should be cleared when the circuit is initially started. However, it is not the responsibility of the data-path and we will look at initialization in the later labs. Fill in the table to determine how to correctly set the various control signals to perform each operation for the correct functioning (high [1], low[0] or don't care [x]). Synchronous based components (such as registers) should be assigned as high or low in every case, whereas other asynchronous components (such as muxes) should be assigned as don't care when not in use.

Please note that every instruction supported by the processor consists of three non-overlapping stages performed over three cycles (see CPU specification document). The first two stages are common to all the instructions, while the last stage is unique to each instruction. The last three rows in Table 1 correspond to the shared stages of an instruction in the T0 and T1 stages (see CPU specification document), while all the other entries at the top of the table correspond to the execution stage, T2. Table 1 should be filled accordingly which will be also useful to implement the control unit in Lab5. When filling out the top of the table, assume that IR has been loaded with its value during T0.

Note that six multiplexers (four 2-to-1 multiplexers and two 3-to-1 multiplexer) are present in the data-path of Figure 1 and they are presented as separate components. However, this need not be the case; you are free to implement the multiplexers either as if-else statements inside your main VHDL file, or you may create a separate component (VHDL file) for the multiplexers (as you did for the ALU, PC and registers) and then instantiate these multiplexers into your design.

3.1 Memory-based Operations

For LDA/ LDB, the value to be loaded is available at the **output** of the **data memory module**. Thus MEM_IN, MEM_OUT and MEM_ADDR will need to be set accordingly for debugging purposes in your VHDL. The values found in IR are provided by the **instruction memory.** Since instruction memory will be created in a later lab, we will use the DATA_IN signal to input instructions into the data-path for this lab. The final lab will be responsible for mapping DATA_IN to the instruction memory output.

For the LDA, LDB, STA, and STB operations, note that IR must first be latched. The latched IR[7..0] address can then be sent to the data memory simultaneously with the data from/to register A or B. This is also the case for other IMM and ADDRS based instructions. For the LDAI and LDBI instructions, A and B will be loaded respectively with the immediate values found in IR passing through the LZE.

Referring to the CPU Specification document (specifically the ASM and memory charts) we observe that the wen and en signals must be asserted in T1 for LDA, LDB, STA, and STB so that they may finish execution in T2. In addition, the STA and STB mux signals must also be asserted within T1 to guarantee that the data can pass through the muxes and be stored by stage T2. Therefore when creating the waveform simulations for these instructions, ensure that the value has been latched by IR first, and that the correct memory enable signals are asserted thereafter. Ensure that you take setup and hold times into account, and that the frequency of operation is set appropriately to avoid ambiguity between testing errors versus implementation errors.

3.2 Zero Extenders (UZE & LZE)

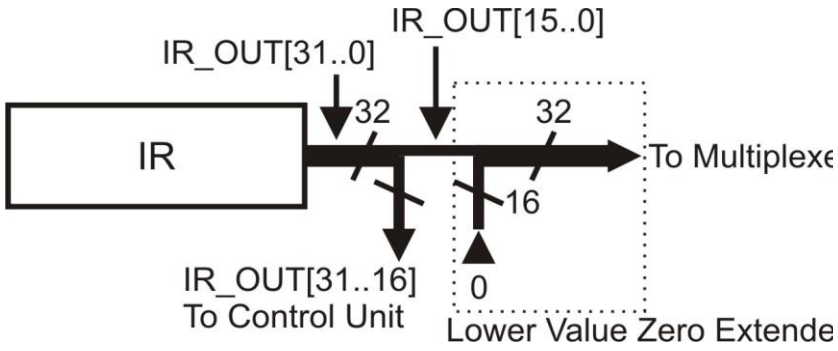

Figure 4: Lower Value Zero Extender

The CPU datapath makes use of two types of zero extenders, for immediate operations and for loading addresses into the Instruction Register (IR). In the case of Load Upper Immediate (LUI), the lower 16-bits of the IR are loaded into the upper 16-bits of register A (i.e. A[31..16]). Therefore UZE must be used that places the LSB16-bits of IR as the MSB 16-bits at the destination. To implement LUI, LZE is used a s the first i nput of the ALU while clearing B, adding the two values together, and storing the final result in A. For the other immediate operations (ADDI, ORI), the lower 16 bits of the IR are used as the second operand, and make up the lower portion of the operand (bits 15..0). In all the cases, the remaining 16 bits are filled with 0s. Students should consult the data-path of Figure 1 and the *CPU Specification* document for implementing these particular instructions.

A zero extender is constructed by segmenting a bus, and selectively grouping wires together. Figure 4 provides an example of a Lower Value Zero Extender (one that keeps the lower 16 bits, and replaces the upper 16 bits with 0).

3.2 Reducer (RED)

To perform load and store operations, the RED unit simply takes the 32-bits from the IR register and outputs the lower 8-bits of the IR. These 8-bits are then inputted into the address port of the data memory unit. Since the addr_in port of the "data_mem" component is of type UNSIGNED, a conversion must be made (since IR is of type STD_LOGIC_VECTOR). This is typically done with the VHDL casting statement as follows:

unsigned_sig <= unsigned(stdlogic_vector_signal)

4 What to Hand In:

To receive full credit for the data-path design, students must submit the following:

- Submit a hard-copy of your data-path code (with adequate documentation) and timing simulations for all instructions given in the CPU Specification document.

- Submit a hard copy of the completed table of control signals (next page Table 1). You only need to provide the single page for this: write your name, student number, and lab section on the table. When filling out the top of the table, assume that T0 and T1 have already executed (i.e. the instruction has been placed in IR and the PC has incremented, i.e. IR <= M[INST] etc). For LDA, LDB, STA and STB, fill out the table according to the signals needed for both T1 and T2, while disregarding PC <= PC + 4 for T1.

- Answer the following questions in a comment section at the END of your data-path VHDL program:
  - How does this data-path implement the INCA, ADDI, LDBI and LDA operations?
  - The data-path has a maximum reliable operating speed (Clk). What determines this speed? (i.e. how would you estimate the data-path circuit clock)?
  - What is a reliable limit for your data-path clock?

Your lab supervisor may also quiz you at the time of the demo regarding the implementation of your CPU data-path.

Student Name:_____ Student#:_____ Section: _____

INST	CLR_IR LD_IR	LD_PC INC_PC	CLR_A LD_A	CLR_B LD_B	CLR_C LD_C	CLR_Z LD_Z	ALU OP	EN WEN	A/B MUX	REG MUX	Data MUX	IM_MUX1 IM_MUX2
LDA												
LDB												
STA												
STB												
JMP												
LDAI												
LDBI												
LUI												
ANDI												
DECA												
ADD												
SUB												
INCA												
AND												
ADDI												
ORI												
ROL												
ROR												
CLRA												
CLRB												
CLRC												
CLRZ												
PC <= PC+4												
IR <= M[INST]												
PC <= IR[15..0]												

Table 1: Data-Path Control Signals