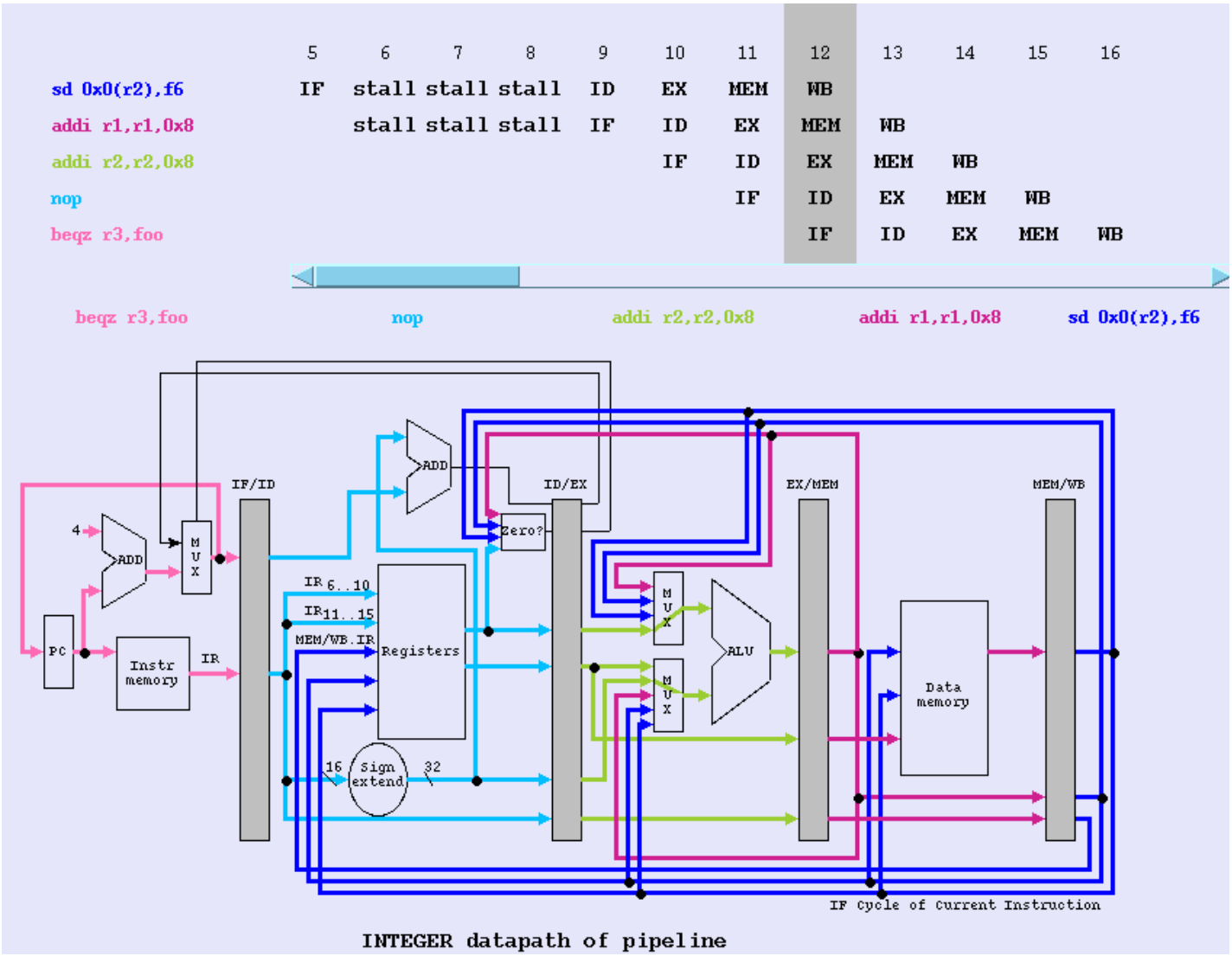
DLX Verilog Design -- Fall 2021

Laboratory Exercise 4

In this assignment, you are asked to complete the design of a RISC microprocessor called the DLX. The processor specifications are taken from the book *Computer Architecture: A Quantitative Approach* by David A. Patterson and John L. Hennessy. You are asked to design two of the pipeline blocks at the HDL level. These blocks would then be combined with the code for the rest of the chip (provided to you) and the entire design needs to be simulated, and debugged.

A block diagram of the DLX processor is shown here. The block diagram also traces out the flow through the pipeline of the different instructions. Do note that this figure shows the data and instruction memory as part of the design. The design you are given to start with does not include memory. These will be provided later in a test bench design into which you can plug in your design before you start simulation.



INTEGER datapath of pipeline

I. Brief description of DLX

DLX is a five-stage pipelined processor. The five stages are,
1. Instruction fetch (IF)
2. Instruction decode (ID)

3. Execution (EX)
4. Memory access (ME)
5. Register write back (WB).

Whenever there is a jump, branch or data dependency, this pipeline will stall at the instruction fetch stage until the last three stages of the pipeline are empty.

The events happening at every stage are shown in the table below.

Stage	ALU Instruction	Load or store instruction	Branch Instruction
IF	MAR ← PC; IR ← Mem[MAR]; PC ← PC+4;	MAR ← PC; IR ← Mem[MAR]; PC ← PC+4;	MAR ← PC; IR ← Mem[MAR]; PC ← PC+4;
ID	$ALUinput_A \leftarrow (R_{S1}, PC)$; $ALUinput_B \leftarrow$ $(R_{S2}, (IR_{16})^{16}\#\#IR_{16..31})$;	$ALUinput_A \leftarrow (R_{S1}, PC)$; $ALUinput_B \leftarrow$ $(R_{S2}, (IR_{16})^{16}\#\#IR_{16..31})$;	$ALUinput_A \leftarrow (R_{S1}, PC)$; $ALUinput_B \leftarrow$ $(R_{S2}, (IR_{16})^{16}\#\#IR_{16..31})$;
EX	$ALU_{output} \leftarrow R_{S1}$ *op* R_{S2}; *or* ALUoutput ← R_{S1} *op* $((IR_{16})^{16}\#\#IR_{16..31})$;	MAR ← R_{S1}+ $((IR_{16})^{16}\#\#IR_{16..31})$; MDR ← R_D	ALUoutput ← PC + $((IR_{16})^{16}\#\#IR_{16..31})$; cond ← ($R_{S1}$ *op* 0);
MEM		MDR ← Mem[MAR]; *or* Mem[MAR] ← MDR;	if(cond) PC ← ALUoutput;
WB	$R_D \leftarrow ALUoutput$	$R_D \leftarrow MDR$;	

The inputs/outputs/registers needed at every stage are shown below.

```
1. Instruction fetch (IF)
     Issue addresses and fetch instructions from instruction memory.
     Pin explanation:
INPUT-
        IR2:            Instruction that is being decoded in Stage II
        Equal:          Result from the Equal Comparator in stage II
        MRST            Reset signal from program counter.
OUTPUT-
        PCMuxSelect:    Selects signal for the Program Counter Mux in stage I
        PCVector:       Exception Jump Vector
REGISTERS-
        You need two registers :- a two bit register which feeds the
        PCMuxSelect signal and a one bit register which is needed to
        determine if a branch was taken in the previous clock

        Notice! Instruction memory is not provided at this time. This will
             given to you later as part of the test bench.
```

2. **Instruction decode (ID)** Get instructions from instruction memory and decode instructions. Pin explanation: INPUT- IR2: Instruction of Stage II OUTPUT- PCAddMuxSelect: Select signals for the PCAddMux in Stage II 3. **Exec Control:** Execute alu functions. Store data to data memory if necessary. Decode opcode and set controls for MUXes feeding ALU etc. The Following are the inputs and outputs of the Execution control: Inputs

- IR3: Instruction feeding the third stage of the pipeline
- IR4: Instruction feeding the fourth stage of the pipeline
- IR5: Instruction feeding the fifth stage of the pipeline
- Shiftamount: In case of a shift operation -> gives the amount by which a shift needs to be performed

Outputs:

- DestMuxSelect: The destination mux select acts as the control for the destination mux. It is a 2 bit control for a mux which switches five bits of data. The mux decides where the output of the ALU should be sent to. It directs it either to the target register (of current instruction), destination register, register 31, or to a flip flop.
- DataWriteMuxSelect : Selects either the source or the data output of DLX.
- ALUSelect : 6bit , selects the operation that the ALU is required to perform.
- Shift select : 5bit, gives the amount of shift the shifter needs to shift the data by.
- ALUorShiftMuxselect: Selects whether the ALU or the Shifter need to be used for the current operation.
- SourceMuxSelect: There are two muxes feeding the ALU- the source mux and the target mux. The source mux select acts as a two bit control for the source mux. The source mux selects between the output of the WriteBack mux, the OUT1 of the register file, the data address o/p for the data cache (useful in load instructions to calculate the data address) , and the Program Counter address.
- TargetMuxSelect: This acts as a two bit control for the target mux. The target mux selects between WriteBack output, the OUT1 of the register file, the sign extended immediate data, and the data address output for the cache.
- CompareMux1select: Does the same job as the SourceMuxSelect but for the comparator. (Inputs to the muxes may be different)
- CompareMux2select: Does the same job as the TargetMuxSelect but for the comparator. (Inputs to the muxes may be different)
- CompareResultMuxSelect: Does the same job as the DestinationMuxSelect for the comparator. (Inputs to the muxes may be different)

```
4. Memory access (MEMCtrl)
     Load data from data memory.
INPUTS-
       IR4:    Instruction feeding the fourth stage of the pipeline.
OUTPUTS-
       DRead, Dwrite:  Decides if a read or a write operation is needed.
```

5. **Write Back Control (WBCtrl)** Writes back the results to the register file. INPUTS- IR5 : Instruction feeding the fifth pipeline stage. OUTPUTS- WBMuxSelect: WriteEnable: Disables the write when the registers are not being written to.

Before starting on this assignment you are strongly recommended to

- **Read up on the DLX architecture.**

. Now you are ready to download the designs.

III. Download Information.

For this exercise you are given a partially complete and completely glued together pipelined DLX design. The objective is to open down into the incomplete blocks and complete the design and then to simulate the design using ModelSim.
- **CHANGE** --> To enter code for the two blocks that you need to design, you now need to do the following...

1. Create a new directory (say called DLX)
2. Download this tar-zipped file
3. Unzip/untar it.
4. Now you need to add modules **IDCtrl and IFCtrl** to the dlx_modules.v file. (spell them as shown.).
5. The templates for the modules are :

```
module IFCtrl (
        IR2,                            // Instruction that is being decoded in stage II
```

```
        Equal,                          // Result from the equal comparator in Stage II
        MRST,                           // Reset Signal for CPU
        PCMuxSelect,                    // Select Signals for the PCMux in Stage I
        PCVector                        // Exception Vectors
);

module IDCtrl (
        IR2,                            // Instruction that is being decoded in stage II
        PCAddMuxSelect                  // Select Signals for the PCAddMux in Stage II
);
```

- You will directly invoke the simulator and the synthesis tool. Instructions on this shall be posted below this soon.

Instructions on using the Mentor Graphics hardware description Language simulator.

1. First we look at the setup.
    1. I assume that all your verilog files exist in a directory called "dlx"
    2. You need to create a directory inside this called work In the dlx directory Type in

        ```
        qhlib work
        ```

    3. Now you need to compile all the .v files.
       Type in the foll. Remember to delete the system.v and clkgen.v files

        ```
        vlog ./*.v
        ```

    4. Invoke the simulator with

        ```
        qhsim -c

        OR

        vsim
        ```

    5. The top level design is dlx.v. SElect the module dlx for simulation.

       These files are zipped versions and you will need to unzip and untar these before you can use them.

       Submission and Grading Policy

    6. Testing and submission procedures.