Pipelined Design:

Moving on to the next part of the project now you will implement a 5 stage pipelined architecture for WISC-SP13 ISA. As already discussed in the class the 5 staged pipelined design helps us improve the performance by improving the cycle time. Unlike unpipelined implementation, you will not have a CPI of 1, but the CPI will be greater than 1 due to hazards. Have fun implementing the design.

As usual, all the Verilog coding rules, naming conventions, etc apply to this project also. If you have forgotten them please visit the hw1.pdf (or Canvas-> Files-> Verilog Tools, Scripts & Guides).

Expectations:

Memory:

Same as the unpipelined design you will also use 2 separate memories in this design. The first is for instruction fetch and the second is for data memory. Your memories will be single-cycle perfect memories. You will use the same memory2c.v that you used in the unpipelined design. You should instantiate it twice, once for instruction memory and the other for the data memory (as you did earlier).

5 Staged Design:

The design that you will make will be a 5 stages design similar to the one discussed in the class. So your design will have IF→ID→EX→MEM→WB.

Pipeline Hazards:

For the pipelined design you must implement hazard detection to detect the hazards. You also need to implement 2 types of data forwarding which are EX/MEM and EX/WB. As for the other possible data forwarding (like MEM/WB), you can decide whether to implement them or just add a stall.
Lastly, for the branches, you can decide whether to use stalls or make a branch predictor. You can make a simple branch predictor where you always assume not taken branch and when you find that the prediction was wrong you invalidate the older instructions in the pipeline which are incorrect. (by not taken branch I mean nextPC = PC + 2)

Instructions for Extra Credits:

As detailed in the last section of “WiscSP13 - ISA and Microarchitecture.pdf”. You can get extra credits for implementing the SIIC and RTI instructions. If you want to try for extra credits you will

have to implement these 2 instructions in this design. We will be testing these instructions for extra credits.
Note: If you are not going for extra credits you do not need to worry if the tests with SIIC and RTI fail. We will not deduct marks for that.

Testing:

As you may have realized, generating the trace with PC, INSTRUCTION, REG, MEM ADDR, and MEM VALUE is more complicated for a pipelined design than for the single-cycle design. So for the pipelined design, we will opt for a simple trace format - we'll call this the pipe trace and give it a .ptrace extension.

Every cycle that a value is being written to the register file (in write-back stage), or being read from memory (in MEM stage), or being stored to memory (in MEM stage), we will record an entry. PC and the instruction bits will not be recorded. Since NOPs, and branch instruction do not effectively change the registers or memory, they will create no entries in this simplified trace format.

There are 4 differences compared to non-pipelined trace:

1. You must run wsrun.pl with the -pipe option while using this testbench.
2. You will use a new testbench called proc_hier_pbench.v. Our wiscalculator has been modified to generate a compatible .ptrace file also. At the end of a simulation, you will see archsim.ptrace and verilogsim.ptrace.
3. Differences are saved in diff.ptrace. This is an intelligent diff that will annotate the original .ptrace filled with the instructions and the PC, so it's easier to track failures.
4. A new RELAX-PASS status has been added. This status indicates that your processor is doing a set of extra register writes or memory reads or memory writes, but they are not corrupting any state. You should FIX this problem!

Everything else remains the same. To run a program:

```
wsrun.pl -pipe -prog foo.asm proc_hier_pbench *.v
```

On failure, see diff.ptrace. Raw ptrace files are archsim.ptrace and verilogsim.ptrace.

Note: We will use the -list option to run the test cases which by now you should already be familiar with.

Test Cases:

We will run all the test cases from the unpipelined design part and we have added a few more test cases specifically for the pipelined design part.

1. **Simple Tests: (10 points)**

```
wsrun.pl -pipe -list simple.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `simple.summary.log`

Note: If you fail more than 10 simple tests. You will directly lose 75% points for the single-cycle unpipelined project. Make sure you do rigorous testing and ensure that you are not failing the tests. (Also if you are failing the SIIC test and you are not trying for extra credit you can ignore that)

2. **complex tests: (15 points)**

```
wsrun.pl -pipe -list complex.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `complex.summary.log`

3. **rand_simple tests: (15 points)**

```
wsrun.pl -pipe -list rand_simple.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `rand_simple.summary.log`

4. **rand_complex tests: (15 points)**

```
wsrun.pl -pipe -list rand_complex.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `rand_complex.summary.log`

5. **rand_ctrl tests: (15 points)**

```
wsrun.pl -pipe -list rand_ctrl.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `rand_ctrl.summary.log`

6. **rand_mem tests: (15 points)**

```
wsrun.pl -pipe -list rand_mem.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `rand_mem.summary.log`

7. **complex_pipelined tests: (15 points)**

```
wsrun.pl -pipe -list complex_pipelined.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `complex_pipelined.summary.log`

Test Cases for Extra Credits:

If you have implemented the SIIC and RTI instructions you can the following list of tests for testing the instructions.

1. **Extra Credits Tests: (40 points)**

```
wsrun.pl -pipe -list extra_credits.list proc_hier_pbench *.v
```

Rename the generated `summary.log` to `extra_credits.summary.log`
Also, note that you must pass the siic_0.asm in simple tests also.

Write-up detailing your design's hazard management:

You must also submit a document named titled `instruction_timeline.pdf` which gives an explanation of the behavior of your processor for the `perf-test-dep-ldst.asm` test (see complex_pipelined tests). Please use the following format:

Cycle	*Instruction Retired*	*Reason*
1		
2		
etc		

The Instruction Retired field would either be one of the instructions from the test program or a "NOP" if dependencies necessitate any stall cycles. The Reason column should give an explanation of why a stall was needed in that instance.
We are asking for this so that we can understand how you have implemented the design easily.

Additional Notes:

- After running the command check the generated summary.log file. It will show you whether your tests have been successful or not.
- After running the command and successfully passing the test please rename the summary.log file (manually) as requested. You need to submit the renamed file in your final submission.
- Note that if you fail more than 10 tests in the simple test you will automatically lose 75% points. So make sure you test things rigorously.
- Make sure you update the variable names in testbench before you proceed with verifying the code. The variables which you need to change are clearly demarcated in proc_hier_pbench.v.
- Make sure you run the vcheck as well as the name-convention-check script before submitting.

What to submit

You need to submit a single tar file named `project-pipelined.tar`.You will submit it directly on the canvas. It will contain the following files

```
- All *.v files needed to run and verify the implementation
- Updated testbench (proc_hier_pbench.v)
- All *.vcheck.out files
- simple.summary.log
- complex.summary.log
- rand_simple.summary.log
- rand_complex.summary.log
- rand_ctrl.summary.log
- rand_mem.summary.log
- complex_pipelined.summary.log
- extra_credits.summary.log (If you implemented extra credits code)
- instruction_timeline.pdf
```