CS450/650 – W2023 – Verilog Assignment 3

Mark toward final grade:	10%
Due date:	March 22st at 11:59pm
Deliverables:	1. mips.sv 2. A few words (text file is fine) stating the observed speedup for the 3 benchmark programs after pipelining, and the calculated mean speedup 3. (Optional) Any data-path diagrams you drew while working on this assignment
Submission:	Upload to Learn | Submit | Dropbox | Verilog Assignment 3
Late policy:	Please refer to course outline

In this assignment, you're tasked to convert a multi-cycle MIPS processor into a 5-stage pipelined MIPS processor, and observe the performance enhancement from pipelining.

(And since every clock cycle needs to be accounted for in a pipeline, you will also undoubtedly become an expert on the workings of clocked synchronous logic :)

The multi-cycle processor Verilog code that is provided to you includes 11 instructions: ADDIU, SW, LW, J, SLTI, BEQ, BNE, JR, ADDU, SUBU, SLL; and can already execute the benchmark programs: SimpleAdd, SimpleIf, and SumArray.

Thus, the pipelined design should produce the same execution result as the multi-cycle design but in fewer clock cycles. Once your pipeline is rolling, please feel free to implement the simplest way to address data and control hazards to achieve correct execution (hints below).

Hints:

- A register/flipflop is automatically created when you use a non-blocking (<=) assignment inside a clocked always process in Verilog. Please implement pipeline latches this way.
- Instead of a FSM that activates one stage at a time in the multi-cycle design, the pipelined design requires 5 simultaneous stages.
- References on stalling:
  - Pages 8-22 from Lecture 9 "Illustration of pipeline stages_stall_fwd.pptx".
  - Page 19 from slide deck "CS450-Ch4-pipelining.pdf"
  - Recall from CS251, one way to squash an instruction is to change its opcode to a NOP and clear its control signals.
- Once your pipeline can squash instructions, you can also use this logic to handle control hazards caused by Jumps and Branches.
  - Page 39-49 from Lecture 9 slides "Illustration of pipeline stages_stall_fwd.pptx"

Note: Whether to support a branch delay slot is your decision. There is no impact on the 3 benchmark programs because they all have a nop in the delay slot.

Relevant files:

MIPS processor	
mips_base.sv -- Please rename to mips.sv	The multi-cycle processor Verilog code for you to modify
Related modules	
memory.sv	Memory module This memory module has registered outputs! 1) When you use it in a pipeline, the pipeline latch/register that normally captures the output from instruction and data memory is already included within the memory module! 2) For A3, we have added 2 extra control signals to this module: **stall** and **clear**. You can use these to prevent its output register from updating, or to force the register value to 32'd0, respectively.
regfile.sv	Register-file module For A3, we have modified this register-file module to update its contents in the first half of the clock cycle. This means you can read out the newest value of a register that is being written to in the same clock cycle (as discussed in lectures).
params.sv	Memory and processor initialization parameters
Testbench file	
a3_tb.sv	Contains several testbenches -- for different benchmark programs and debug versions that give detailed printouts. Each testbench instantiates MIPS and memory modules, loads a benchmark program, and prints the execution outcome.
Benchmark programs	
SimpleAdd	.c — Original source code .dmp — Listing generated by MIPS compiler .s — Assembly code generated by MIPS compiler .sh — Compile script .x — Object code in hexadecimal format – goes in the same folder as .sv files
SimpleIf	ditto
SumArray	ditto

Commands to compile, simulate, and view waveform

```
iverilog -g2005-sv -s tb_SumArray_debug a3_tb.sv memory.sv regfile.sv mips.sv

vvp -n a.out

gtkwave dump.vcd
```

Note: Which testbench to compile is specified in the iverilog call