Brown University
School of Engineering
EN164 Design of Computing Systems
Professor Sherief Reda
Assignment 07 (150 points)
Final lab due on Friday, April 19 (Milestone on April 12)

BROWN

In this lab, you are required to substantially improve your original single-cycle processor design by using pipelining. You will need to submit only one zip file (lab report + qar file) on the due date of April 19. However, you will need to demonstrate progress by submitting a zipped .qar file on April 12th as outlined below.

Guidelines:

- Similar to the single-cycle processor, to verify that your processor works correctly, you need to store the execution results in the RAM data memory and then read the memory contents with the Quartus II tool after you run and are done with the program.
- The design metric for this lab is execution runtime, which is equal to the number of actual clock cycles taken by the program multiplied by your clock period. Twenty points of this lab are allocated based on your processor runtime on the factorial (6!) program:
    - If your frequency for the single cycle was 100 MHz (i.e., your runtime was less than 700 ns) then your goal is to improve your improve performance by about 15-20% using pipelining; **otherwise**:
    - single-person team: (20 < 900 ns, 18 < 925 ns, 16 < 1000 ns, 14 < 1025 ns, 12 < 1050 ns, 13 < 1100 ns, 14 < 1125 ns, 6 > 1150 ns).
    - two-person team: (20 < 700 ns, 18 < 725 ns, 16 < 750 ns, 14 < 775 ns, 12 < 800 ns, 10 < 825 ns, 8 < 850 ns, 6 > 875 ns).
    - If your pipelined design ends up slower than your single-cycle design, you get zero points in the entire assignment!
- To count the number of clock cycles, you should keep a running count of the number of executed cycles in a register (let's use x30). That is, you need to increment this register every clock cycle, and store the value of this register in memory before you halt., e.g., add this instruction: sw x30, 4(x0) before the halt in the factorial program.
- It is probable that the EX more than 50% of the delay of the critical path in your single-cycle design. Since the frequency of your pipelined processor is determined by the stage that has the highest delay, the EX stage may determine the maximum clock frequency of your pipeline processor. Thus, there may be no benefit of having separate MEM and WB stages, and it is better to combine them into one stage. So, you might like to pursue a two-stage pipeline processor (IF-ID-EX / MEM-WB) or a three-stage pipeline processor (IF-ID / EX / MEM-WB). Both designs are acceptable, and it is up to you to choose. The two-stage design will simplify hazard detection and forwarding. If you want to get a big speed-up, you can pipeline your

multiplier (check Megawizard) effectively splitting the bottleneck EX stage into two or more stages (e.g., EX1 and EX2).

- You need to think of interesting ways to clock your data memory in your pipeline design to avoid unnecessary penalties, and we discussed several strategies in the context of the single-cycle processor design.
- Please verify that your design works using the same program (factorial and your program of choice) as you did in Assignment 06.

(a) Due Date April 12. Implement the pipelined processor assuming no hazards exist in code. To run the factorial program correctly, insert **nop (e.g., add x0, x0, x0)** manually in your code to eliminate all hazards.

(b) Due Date April 19. In this case, you need to modify your pipeline design so that it stalls whenever it detects a hazard. Verify that you execute the same number of cycles as in part (a). Since you stall in hardware, you can remove the inserted **nop**s.

(c) Due Date April 19. You need to demonstrate the final version of your design where it forwards operands to avoid RAW hazards whenever possible but stalls otherwise. With forwarding implemented, you can remove the **nop** instructions in the factorial code. What is the new number of cycles that the program takes to finish executing?

<u>Use part (c) as the reference for the following requirements of your report:</u>

1. Include the assembly and machine code of the factorial program and your program of choice in the documentation. **Make sure that your program of choice can detect both RAW load-use and compute-use hazards.** Print screenshots that show the memory contents after executing the factorial program and your program of choice. **Analyze your programs and prove that the reported numbers of cycles are correct.**

2. Use the TimingQuest tool to analyze the timing in your design. Report the critical path delay in your design and predicted Fmax. Use the tool to (1) locate the critical path in your floorplan and print the annotated critical in the floorplan view, and (2) estimate the delay breakdown among the different stages of your pipeline. **Explain how you used the timing path information to optimize your design so that it can run at a higher speed. Explore the possible optimization settings of the Quartus II tool and report the impact of relevant different settings on the timing and area of your design.**

3. Using the actual board, find the maximum frequency that your processor can sustain without producing incorrect results. You need to keep on incrementing the frequency of the design by increasing the PLL multiplier,, and re-running your experiments. Once the processor fails in booting, report the lowest frequency in which such failure occurs. Contrast the actual maximum frequency to the estimated Fmax from the TimingQuest tool. Why is your reported max frequency higher than the one reported by the tool

4. Explain your branch resolving strategy.

5. Print and annotate the floorplan of your design. Report the resources being used by your processor: LEs (combinational and dedicated registers), PLL, embedded multipliers, memory blocks, and routing resources.