Lecture #20 Worksheet, Answer Master

Fill in blanks to answer questions below. Then email this sheet to your TA.

1. **Although the MIPS multicycle implementation provides a substantial 30-40% speed improvement, it has the disadvantage that the five segments of the processor must be completed to finish an instruction. When one segment is active, what are the other four segments doing?**

   Nothing. The other four segments are idle.

2. **Patterson and Hennessy provide the "laundry example" to illustrate how parts of the processor might be simultaneously active. Explain this example.**

   Basically, the processor is redesigned to simultaneously "washy, dry, fold, and put away" instructions. I.e., more than one instruction is processed at once.

3. **Slides 4-6 illustrate the basic idea of the pipeline. Explain in your own words.**

   The pipeline is to be redesigned so that every time the clock ticks, another instruction is fed into the first segment. Thus instructions "march" down the pipeline, every segment processing part of an instruction every cycle.

4. **Why does the speed advantage of the pipeline approach s (s the number of stages in the pipeline)?**

   The ratio of the multicycle speed (ET_S) to the pipeline speed (ET_P) is: $S_P = \frac{ET_S}{ET_P} = \frac{ns}{s+(n-1)} \rightarrow \frac{ns}{n} \rightarrow s$**, for large *n*. Thus the pipeline increases processing speed by a huge factor.**

5. **State the five stages of the pipeline, which correspond to the five stages of the multicycle implementation.**

   The stages are IF (instruction fetch), ID/RF (instruction decode/register fetch, ALU (ALU cycle, sometimes called the execution [EX] cycle), MEM (memory access cycle), and WB (writeback to register block cycle).

6. **Slide 9 shows the overlapping of the instruction cycles. How many instructions**

are being simultaneously processed when the pipeline is full?

Five (5).

7. **Going back to the single cycle implementation, we see that once again, there is a need to save partial results as instructions go down the pipeline. How does this compare to the multicycle implementation?**

   Partial results must be saved in the same four places for either.

8. **Slide 11 shows the rudimentary pipeline (no control lines) with storage. What are the acronyms for the four interfaces?**

   They are IF/ID, ID/EX, EX/MEM, and MEM/WB.

9. **Slides 12-16 show the progress of an instruction down the pipeline. Study these carefully. Why is the program counter contents carried along?**

   In some cases, the PC must be modified, as for a jump or branch, and this is done later in the instruction.

10. **Slide 18 shows the pipeline with all control lines added (the jump circuitry, which is fairly simple, is not included for clarity). Questions:**
    a. **Why are the identities of the registers carried along on the register address bus?**

       The register ID's are not used until later cycles.

    b. **Control signals are also taken down the pipeline. Why?**

       For the same reason—needed in later cycles.

    c. **What does the lower MUX in the ALU (EX) cycle do?**

       It chooses between the Rt field (load instructions) and the Rd field (R-R instructions) for the destination register.

11. **Slides 19-31 demonstrate the transition of five instructions down the pipeline. Study these carefully to understand how instructions are processed. In the multicycle implementation, only those sections of the instructions that are**

needed are included. Is this true in the pipeline?

No. Every instruction goes through all five sections of the pipeline.

12. Complete the exercise on slide 32, using the diagram on slide 33. The answers are shown on slide 34.

13. Hazards are a problem due to pipeline design that can result in the wrong data being accessed. The two types of hazards are discussed in slides 35-38. Although differently named, control hazards (in branches) and data hazards (as in register instructions) both result from exactly the same pipeline problem. Explain this problem in your own words.

A hazard results when an instruction accesses register contents of a register which is about to be reloaded by an instruction currently in the pipeline. That is, the register is a destination register of an instruction currently in the pipeline. This means that the instruction accessing the current data will get the wrong data (the register contents that will be changed in 1-3 cycles).

14. How does forwarding solve the hazard problem?

The forwarding unit takes the result of the ALU processing that will result in the change of a register value and sends (*forwards*) it directly to the register, substituting the new value for the incorrect value still in the register.

15. Study slide 40.How does the forwarding unit decide when forwarding is required?

Destination register ID's on the register address bus are compared to source register ID's in instructions being decoded. Where there is a match, the destination register contents are substituted for the older data being accessed from the register prior to its being changed.

16. As slides 42-44 explain, sometimes forwarding will not work by itself. When does this occur and how does the pipeline process address this problem?

If a register is about to receive new data from memory, the ALU output is an address, not the required data. The pipeline therefore pause the two upstream instructions (a stall) while letting the three downstream instructions in the pipeline continue to progress. When the downstream instruction receives

memory data, that data can then be forwarded to the upstream instruction and substituted for the incorrect data still in the register.

17. **Study slides 45-48. Branching-related problems are due to the fact that in the unaltered pipeline, the branch decision is made in the ALU or EX cycle. With no way of identifying the branch results early, two incorrect instructions may have been loaded into the pipeline. Name three ways that this problem can be addressed.**

    a. **A simple solution is to always assume the branch is or is not taken. Therefore, if assuming "taken," always load the next instruction from the branch route. If this is the incorrect decision, two instructions must be "flushed."**
    b. **A second solution is to put the branch detection in the second stage, resulting in having only one instruction to potentially flush.**
    c. **A third approach is to keep a "branch history." A branch flip flop is set if branch is taken, reset if not. The instruction load circuitry checks the branch bit and loads the following instructions based on the branch bit.**

18. **What two improvements can be made on these basic methods?**

    a. **If two branch bits are used, the pipeline will set branch or no branch when the bits agree. If they differ (has been branching but suddenly didn't branch, or vice-versa), it checks history and does what it has been doing most frequently in recent history.**
    b. **Combining early branch detection plus branch history can lead to very high accuracy in predicting branch paths, increasing pipeline efficiency.**

19. **Answer the questions on slide 49, then check your answers on the last two slides (50-51).**

20. **Study the summary slide on branching. Remember: performance is bought with $$$ and complexity. The MIPS pipeline is a good example of both.**