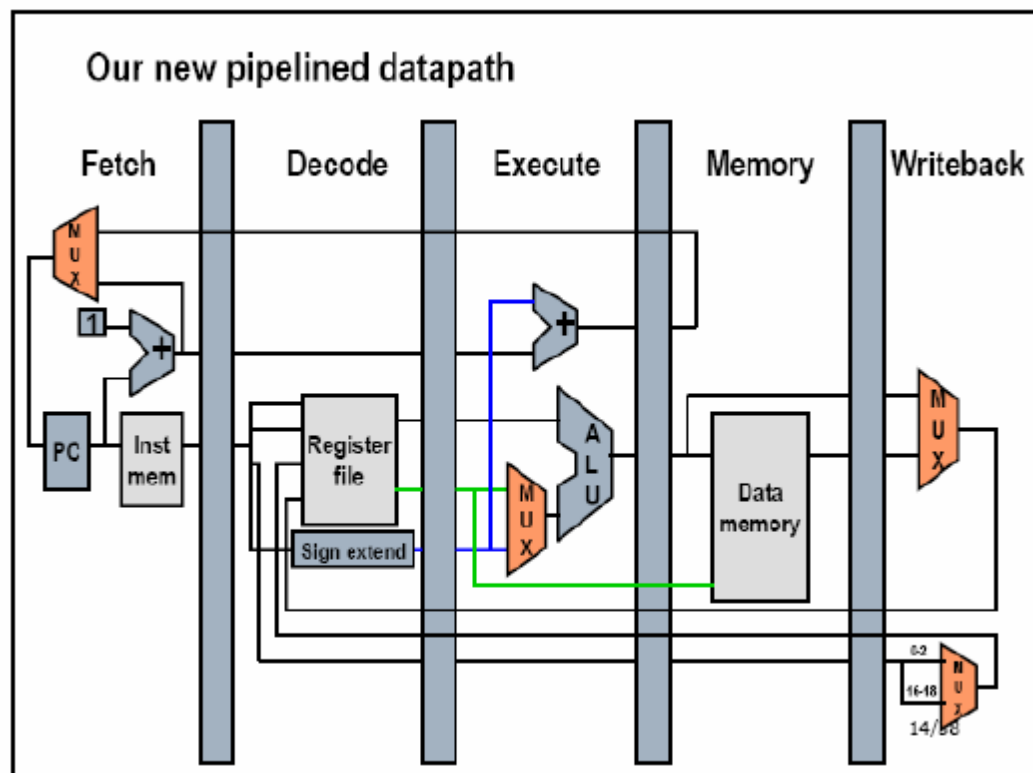


EECS 370

Discussion 6

Week of 3/6 – 3/12

- I) Overview of Datapaths
 - a. Single Cycle
 - i. $CPI = 1$
 - ii. Clock cycle = slow
 - b. Multi-Cycle
 - i. $CPI > 1$
 - ii. Clock cycle = fast
 - c. Pipelining
 - i. $CPI \sim 1$
 - ii. Clock cycle = fast
 - iii. Decreases CPI while not impacting clock period!
- II) Basic pipelining



- a. Added special pipeline registers between stages

- III) Hazards
 - a. Data hazards
 - i. “Ostrich in the sand” approach: Avoidance
 - 1. Rely on the assembly code to be hazard-free
 - 2. Noops inserted to avoid hazards
 - ii. Detect and Stall
 - 1. If a hazard is encountered, stall the processor until they go away
 - 2. Compare your destReg with regA/regB of newer instructions
 - 3. Like avoidance, stall with noops, but done in hardware instead: insert noops into pipeline
 - iii. Detect and Forward
 - 1. Detect like in detect and stall
 - 2. Use **bypass datapaths** to redirect freshly computed data to where it’s needed
 - 3. You may still need to stall! ☹
- IV) Examples
 - a. Where are the data hazards?
 - 1. Add 1 2 3
 - 2. Add 3 4 1
 - 3. Nand 3 2 6
 - 4. Nand 5 2 7
 - 5. Add 3 5 1
 - 6. Lw 6 8 100
 - 7. Add 8 7 2
 - 8. Halt
 - b. How can we resolve them? (using detect and forward)
 - c. How many total cycles?
- V) Project 3 Hints
 - a. Differences between Project 3 pipeline and lecture/book pipeline
 - i. Less hand-waving: cannot read and write from same register/memory in same cycle
 - ii. Additional pipeline register after writeback stage: makes it more convenient for you
 - b. Implement it WITHOUT hazards first... or you may regret it
 - c. Add in hazards only after your implementation is fully debugged
 - d. Likely the hardest project: start EARLY
 - i. Testing this project is much trickier
 - ii. Think about all possible scenarios involving execution and hazards
 - iii. Randomly hacking test assembly programs may not work so well this time!