CSE372
Digital Systems Organization and Design Lab

Prof. Milo Martin

Unit 7: Hints on Pipelining & Wrapup

Agenda

- Ramblings on design & testing
- Discuss pipelining issues
  - Some (hopefully) helpful hints
  - BlockRAM troubles
- Discuss where CSE371/372 should go in the future
- Course evaluations

Optimism

- "We're almost done, we just have to test it."
- From Fred Brooks' *The Mythical Man-Month*:

Optimism

All programmers are optimists. Perhaps this modern sorcery especially attracts those who believe in happy endings and fairy godmothers. Perhaps the hundreds of nitty frustrations drive away all but those who habitually focus on the end goal. Perhaps it is merely that computers are young, programmers are younger, and the young are always optimists. But however the selection process works, the result is indisputable: "This time it will surely run," or "I just found the last bug."

Testing and Testbenches

- "Good Enough"
  - On an exam, 95% is a good score
  - In "design", 95% correct isn't good enough
  - Different mentality
- Testbenches are not just academic artifacts for grading
  - Real systems use "unit tests" and randomized testing to find bugs
- Testing is integral to any development project
  - For a three-week (almost four) project, how much of that should be testing?

More Fred Brooks

For some years I have been successfully using the following rule of thumb for scheduling a software task:

⅓ planning
⅙ coding
¼ component test and early system test
¼ system test, all components in hand.

This differs from conventional scheduling in several important ways:

1. The fraction devoted to planning is larger than normal. Even so, it is barely enough to produce a detailed and solid specification, and not enough to include research or exploration of totally new techniques.
2. The *half* of the schedule devoted to debugging of completed code is much larger than normal.
3. The part that is easy to estimate, i.e., coding, is given only one-sixth of the schedule.

More Fred Brooks

In examining conventionally scheduled projects, I have found that few allowed one-half of the projected schedule for testing, but that most did indeed spend half of the actual schedule for that purpose. Many of these were on schedule until and except in system testing.[2]

Failure to allow enough time for system test, in particular, is peculiarly disastrous. Since the delay comes at the end of the schedule, no one is aware of schedule trouble until almost the delivery date. Bad news, late and without warning, is unsettling to customers and to managers.

Design

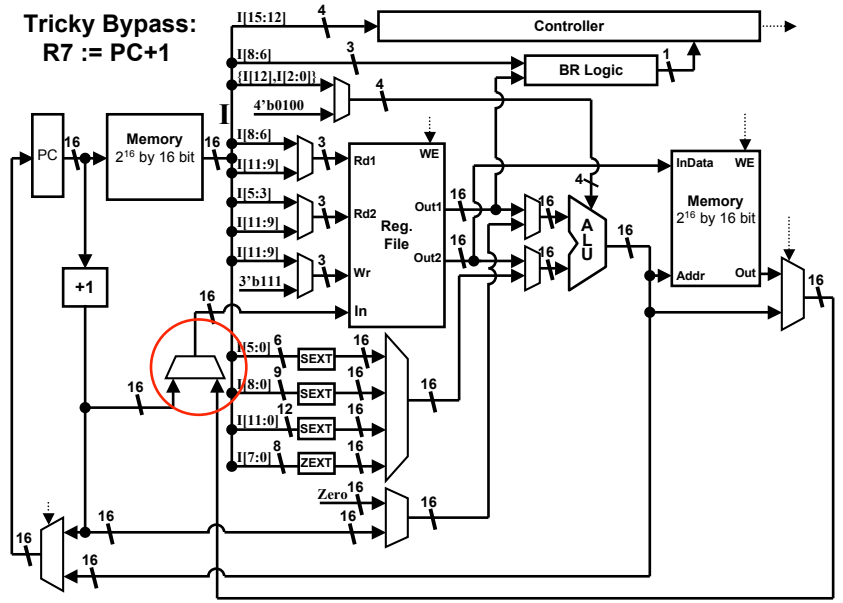
- Design matters
  - Getting this working isn't just "implementation", it requires design
  - A strong design makes lots of difference
  - This project is too difficult to brute force
- Can't take the CSE371 slides too literally
  - Design to explain pipelining, not an actual implementation
- Few discussed implementation of bypassing and stalling in design document

Bypassing and Stalling

- Fully decode instruction vs latching it each cycle
  - Think about how the CSE371 homework abstracted this issue
- For each instruction:
  - Determine what register it writes
    - 3-bit register ID, 1-bit "write enable" valid bit
  - Determine what register it reads
    - Two x (3-bit register ID, 1-bit "read enable" valid bit)
  - Does it write memory? (just the "write enable")
  - Does it read memory?
- Once you have this, bypassing and stalling should be mostly opcode and instruction independent

Some Tricky Bypassing Cases

- LDR r2 ← [r1+10]
  STR r2 → [r3+5]
- JSR LABEL
  LABEL: ADD R0 ← R7, R0
- Note: be sure to "next-PC" predict all sorts of control transfer instructions
  - In fact, just predict "all" instructions, should work just fine

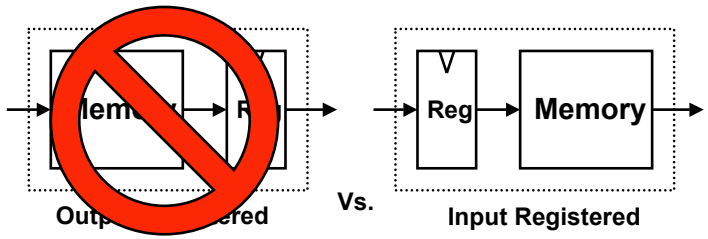


Nullifying Instructions

- How to squash an instruction?
  - Approach #1: mux in a NOOP encoding
  - Approach #2: set an explicit "not valid" bit
  - Approach #3: set all "read enables" and "write enables" to zero
- My suggestion: some combination of approach #2 and #3
  - Goes along with not tracking the actual instruction encoding everywhere
- Note: need to track type of stall or squash for performance counters anyway…

Block RAM Troubles

- Our 128KByte memory must use Xilinx "block RAMs"
  - Wouldn't fit on the FPGA otherwise
- Xilinx blockRAMs are synchronous read
  - Unlike our asynchronous read register file
  - Hard, real-world constraint; we need to work around this



Block RAM Partial Solutions

- Approach #1: Use the global write enable (GWE)
  - Use it to make the BlockRAMs look asynchronous
  - Add explicit pipelined registers where needed
- Approach #2: Stay the course
  - Keep with assumption that BRAMs are input registered
  - Handle some of the tricky stall cases by changing read address
- Approach #3: Add read enable to BRAM
  - Like approach #2, but simplifies stall logic (See TAs for code)
- Approach #4: Assume output registered BRAMs
  - Unfortunately, hard to do bypassing into Memory stage

Register File Bypass

- Our register file hands writes differently than book
  - Solution: add one more local bypass
  - Can be done totally internal to register file
- Why aren't we using both negative & positive clock edges
  - Can really complicate on-board functionality
    - Risk avoidance
  - Disallowed by some standard cell ASIC design flows
  - Should work, but who really knows

Course Recap

- We've talked about ***digital logic design***
  - Verilog
  - Design flows
  - FPGAs and hardware devices
- We've talked about ***design***
  - Breaking a task into parts
  - The process of design
  - Hands on experience
  - Learning by doing
- Recall: last year, no CSE372 lectures
  - They were on their own

CSE372 in the Future

- What should be do next year?
  - Same as this year (1.0/0.5 credit split with separate lab lecture)
  - Combine CSE371/CSE372 into a single class
    - Remove some of the material
    - Keep project
  - Abandon project altogether (no, in my opinion)
  - Split into two 1.0 courses in different semesters
- Your thoughts?