- CSR Instructions
    - csrw, crswi
- Pipelining synchronous r/w
- Resolve hazards
    - Read-after-write data hazards
    - Control hazards
- Reg File
- RAMS
- BIOS
- Address Space Partition
    - Giant block of memory
    - Split permissions based on address input
- Mem-Mapped I/O
    - Memory space allotted for UART signals and CPI arithmetic
    - Software handled: uart_rx_data_out_valid and uart_tx_data_in_ready
    - CPU handled: uart_rx_data_out_ready and uart_tx_data_in_valid

Timeline

- ALU/Imm Gen/Branch Comp/Control logic w/o forwarding (1 week) (Nov 11)
- BIOS/Memory (2 days) (Nov 13)
- Pipeline/Predictor (5 days) (Now 18)
- UART (3 days) (Nov 16)

EECS151

- Registers

Checkoff notes:

- UART: memory mapped addresses sends read and write to UART
- PC reg extra cycle
- Extra forwarding because of longer ID critical path
- S

Forwarding:

- Add register on forwarding
- Mem-aLU might be longer critical path
- Special case: grab wb value before clk writes back (depends on implementation)
    - Reg in decode stage causes this
- Critical path could be in WB/forwarding paths
- UART consider store/read instruction

1. How many stages is the datapath you've drawn? (i.e. How many cycles does it take to execute 1 instruction?)
    a. 3 Stages
2. How do you handle ALU → ALU hazards?
    a. Forwarding: We forward the result of the ALU back into AMUX and add forwarding control signal to AMUX. Forward if inst0[rd] == inst1[rs1] and/or inst2[rs2]
3. How do you handle ALU → MEM hazards?
    a. Forwarding: We forward the result of the ALU into a new mux controlled by a new forwarding signal that selects between the pipeline register and the ALU result. Forward if inst0[rd] == inst1[rd]
4. How do you handle MEM → ALU hazards?
    a. Forwarding: We forward the result of the MEM stage into AMUX and add forwarding control signal to AMUX. Forward if inst0[rd] == inst1[rs1]
5. How do you handle MEM → MEM hazards?
    a. Forwarding:
    b. Read after write:
    c. if (inst0[rd] == inst1[rd]):
        i. Forward MEM/WB into AMUX
    d. if (inst0[rd] == inst1[rs1]):
        i. Forward MEM/WB into BMUX
6. Do you need special handling for 2 cycle apart hazards?
    a. No special handling given correct forwarding alu to alu
7. How do you handle branch control hazards?
    a. Branches always taken, if prediction is wrong flush
        i. Could progress into last time predictor
8. How do you handle jump control hazards? Consider jal and jalr separately. What optimizations can be made to special-case handle jal?
    a. Jal: flush after decode before Reg[]
    b. Jalr: flush after decode after Reg[]
9. What is the most likely critical path in your design?
    a. Clk-to-q -> RegFile → Branch Comp → MUX →ALU
10. Where do the UART modules, instruction, and cycle counters go? How are you going to drive uart_tx_data_in_valid and uart_rx_data_out_ready (give logic expressions)?
    a. UART (parallel with stage with M/W)
        i. Counters & modules : I/O mem
        ii. assign uart_tx_data_in_valid = ~tx_fifo_empty_delayed; and sw
        iii. assign uart_rx_data_out_ready = ~rx_fifo_full; and lw
    b. Instructions: BIOS mem
11. What is the role of the CSR register? Where does it go?
    a. It is for testing and debugging purposes. It takes in the outputs of rs1 of the Reg[] or immediate generator and is connected to the writeback mux in the WriteBack stage. This goes in parallel with our middle stage

12. When do we read from BIOS for instructions? When do we read from IMem for instructions? How do we switch from BIOS address space to IMem address space? In which case can we write to IMem, and why do we need to write to IMem? How do we know if a memory instruction is intended for DMem or any IO device?
    a. On reset. After BIOS. When BIOS is done with startup. When write to IMEM when in BIOS, and is needed for program binary upload. 4 most significant bits due to data partition.