Lab 3

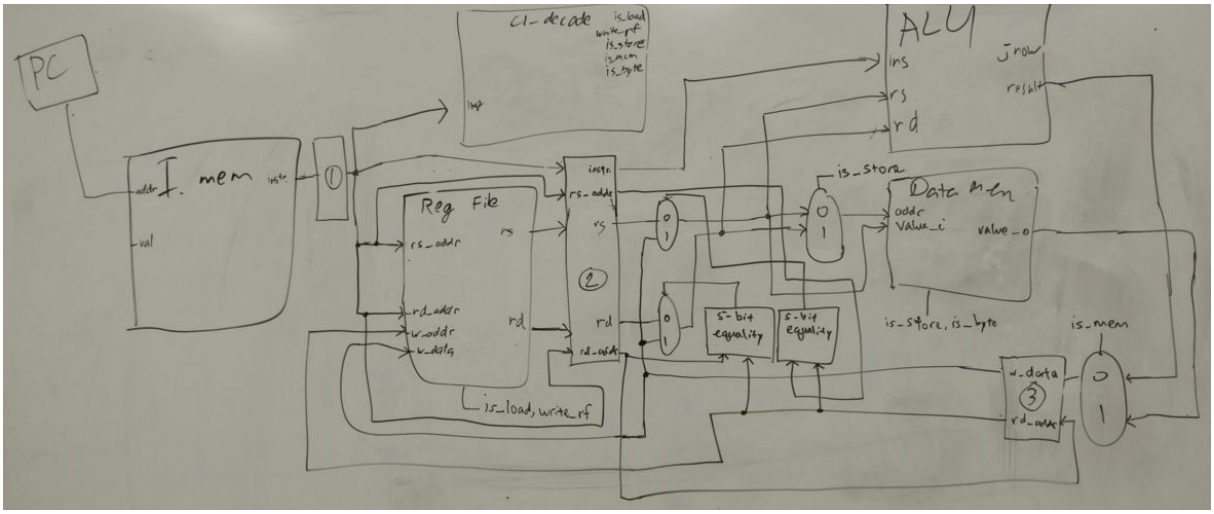
Matt Asaro
Lawrence Lee
Charles McKay

Preliminaries

1. *Perform the following tasks for your non-pipelined processor and include them in your written report:*
    a. *List the reported Fmax of your design before pipelining*
    52.11 MHz
    b. *List the cycle time of your design before pipelining. Show your work*
    1/52.11 MHz = 1/5211000 Hz = **19.2 nanoseconds**
    c. *List the instruction count for miner_tb from the ModelSim simulation before pipelining.*
    120561 instructions
    d. *List the cycle count for miner_tb from the ModelSim simulation before pipelining.*
    131490 cycles
    e. *Compute the CPI and execution time (# of cycles x cycle time) before pipelining.*
    0.00669 seconds
    f. *List the number of registers used, the number of combinational functions used, and the number of memory bits used before pipelining.*
    2069 registers
    4419 combinational functions
    16384 memory bits
    g. *Include a datapath diagram of your core. We strongly encourage you to draw this by hand. It will become a very useful tool in debugging your core.*

2. *Include answers to the following questions in your written report:*
    a. *On your datapath diagram, define the boundaries of stages in your pipeline by drawing in the required registers. Spend some time doing this*

as it will be a useful tool in debugging the core



b. *Discuss the tradeoffs of having a pipeline with more vs. less stages.*
The benefits to having less stages is a less complex processor, fewer potential hazards(which include data dependencies). More stages only benefit if for each cut you're putting it in the section with the longest propagation time. Only then, do you get more throughput because you're processing instructions in parallel. If you place too many pipecuts, there are more data dependencies, and the penalties for it become higher.

3. *Include answers to the following questions in your written report:*
   a. *You will need to pass the proper control signals to each stage of your pipeline. List which control signals are necessary for each stage of your pipeline.*

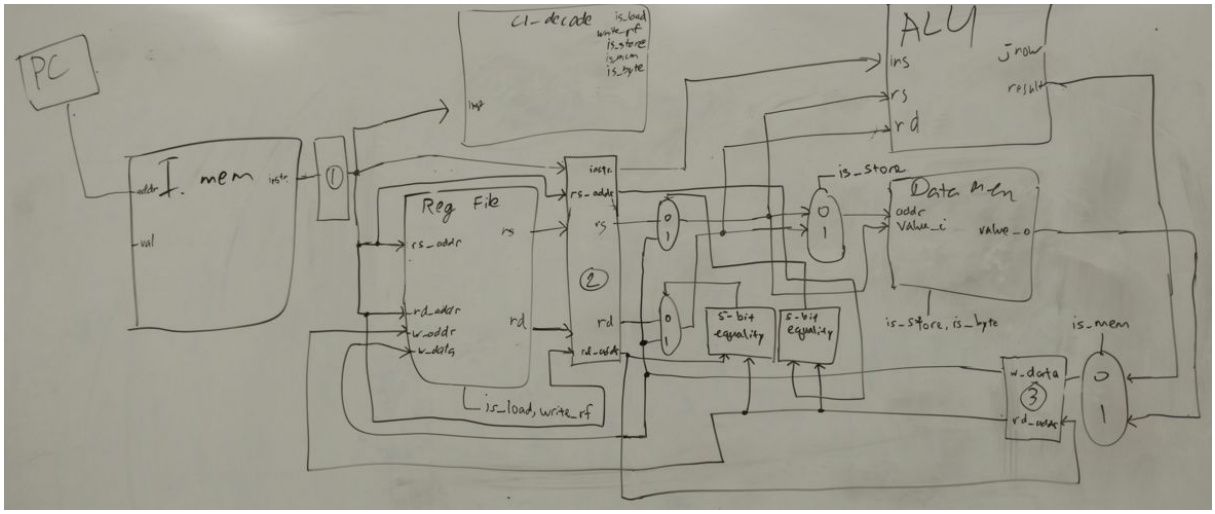
   <u>IF</u>
   No control signals

   <u>ID</u>
   No control signals

   <u>EX/Mem</u>
   is_store_op
   is_byte_op
   is_mem_op
   is_load_op (for stage 4)
   op_writes_rf (for stage 4)

WB
is_load_op
op_writes_rf

b. *Update your datapath diagram to show the additional datapaths, registers, and muxes needed to implement pipeline forwarding.*



4. *Include answers to the following questions in your written report:*
   a. *List all data hazards in your pipeline, and how they should be resolved (which stage data should be forwarded from, etc).*
      rs_addr (stage 3), rd_addr (stage 4), w_addr (stage 4), rs_val (stage 3), rd_val (stage 3), rf_wd (stage 4)
      We will use pipe cuts to forward all data values through pipe cuts until they are no longer needed. Specifically, we will compare the w_addr in stage 4 to stage 3 rd_addr and rs_addr. If either values match, then we will forward and select the rf_wd value instead of the respective stage 3 values.
   b. *List all control hazards in your pipeline, and how they should be resolved.*
      is_store_op (stage 3), is_byte_op (stage 3), is_mem_op (stage 3), is_load_op stage 4), op_writes_rf (stage 4)
      Control hazards are resolved by forwarding control signal values through pipe cuts until they are no longer needed.

   c. *List all structural hazards in your pipeline, and how they should be resolved.*

d. **JALR, BEQZ, BNEQZ, BGTZ, BLTZ**
We will stall new instructions from writing to pipe cuts during the execution of these instructions.

5. *Why would you want to have an independent write address port on the register file for a pipelined processor (i.e. why would you need to write to an address other than the rd address of the instruction in the decode stage)? Hint: think about what instructions may be in the write back stage (the stage which writes to the register file) and the the decode stage (the stage that reads from the register file).*
Because in a pipelined processor the write data is computed in a different cycle than the read. When the write data is ready to be written to the register file the processor is already decoding another instruction which can have different read and write addresses.

Tasks completed:

- The first thing we noticed was that none of our instructions for accessing data memory had any offset. It appeared that the ALU was acting as a mux for the data memory input values. As a result, the ALU and Data Memory will happen in the same stage, in parallel, since the Data Memory is independent of the ALU. This means that we can max out at 3 pipe cuts (4 stages) instead of what is recommended in the lab write up.
- We flush out all pipecuts if we get a reset signal.
- The first pipecut collects the instruction fetched and the PC count of that instruction. It needs the PC for jump/branch calculation. In this first stage, we insert nops if we encounter a jump, branch, or wait instruction. We got this pipecut working 100%.
- We put in the second pipe cut forwarding the instruction, pc, rs and rd address and values. If we calculate a value for a register to be used in the next instruction we write that value into the second pipecut rather than the fetched value of the register. That value is stale. **This is our first use of data forwarding.** However, we currently don't pass the miner test as it finds it on cycle 22.

Tasks Left:

- Finish debugging of Cut 2
- Add Cut 3
- Jump buffer register
- branch prediction
- miner.asm optimizations

Optimization Results:

1. Include answers to the following questions in your written report:
    a. How many stages does your pipelined processor have?
    b. Which pipeline cuts did you successfully implement? Include a diagram.
2. Include answers to the following questions in your written report:
    a. List the reported Fmax of your design before pipelining.
    b. List the cycle time of your design before pipelining. Show your work.
    c. List the instruction count for miner_tb from the ModelSim simulation before pipelining.
    d. List the cycle count for miner_tb from the ModelSim simulation before pipelining.
    e. Compute the CPI and execution time (# of cycles x cycle time) before pipelining.
    f. List the number of registers used, the number of combinational functions used, and the number of memory bits used before pipelining.
    g. What is your speedup (baseline execution time/optimized execution time)?
    h. How does pipelining differ from the optimization method used in Lab 2 (adding instructions to the ISA)? Specifically which of the two methods targets cycle time, instruction count, CPI, etc. Hint: You may want to compare your results from Lab 2 with your results from Lab 3.
    Pipelining increases throughput by decreasing cycle time, but also increases CPI, whereas adding instructions decreases the instruction count.