Marc Slaughter, Nick Otto, Lab2A
3/15/13

1. Since the 'P' bit of the instruction is zero, the instruction is either not a branch or is predicted as branch not taken. Thus the next address in execution will be immediately following 0x12. so 0x13.

2. We assume, since a branch is to be taken, that the 'P' bit is 1. Since the branch is backwards one address, the offset must be negative one, effectively replacing the "normal" pc_r = pc_r + 1 with pc_r = pc_r + offset. Offset, in this case, is -1. -1 in two's complement binary representation is all ones. Thus the instruction will be 1XXXXXXXXXXX11111.

   In this case, our front end datapath will place the instruction from 0x25 in the FIFO queue, then the PC will be updated as described above, finally, the instruction at 0x24 will be fetched and a new branch determination and enqueueing process will begin.

3. The correct address for the fetch unit has two cases.

   Case 1: The most significant bit, the 'P' bit, was zero and a branch was not predicted, but a branch needs to take place. In this case, the execution unit can compute the next address based on the current address and the sign extended offset the same way the fetch unit does.

   Case 2: The 'P' bit was one and a branch was predicted, but a branch should not be taken. In this case, the execution unit can determine the next address by adding one to the current address and sending it back to the fetch unit.

   If the execution unit sets restart_i and restart_addr_i at cycle 10 then the FIFO will be cleared. It will take three cycles for the instruction to make it through the front end unit to the back end. Thus it can begin executing the correct instruction at cycle 13.

4. The pc_r register needs to be split into two cycles so the SRAM knows which instruction to feed into the FIFO and the FIFO itself needs to know the address of the instruction that was previously pulled out of SRAM. The FIFO needs the address (pc_r) as well as the instruction itself so the execution unit knows which address to pass to restart_addr_i if necessary. If this information was not passed to the execution unit, the fetch unit would need to somehow save all of the addresses currently in the FIFO. So, the pc_r register goes both into the FIFO and back around to the beginning of the fetch unit in order to fetch the next instruction.

5. Cycle 0: 0x13, since the address at 0x10 had a P bit of 1 and an offset of 3 Cycle 1: 0x13, since sel_mux[1] (fifo_full) becomes zero Cycle 2: 0x40, since sel_mux[3] (load_store_valid_i) is asserted (The PC still contains 0x13) Cycle 3: 0x13, since the PC maintained its value from before the load operation Cycle 4: 0x50, since sel_mux[3] is asserted (The PC now contains 0x0 from the restart control Cycle 5: 0x0, since sel_mux[1] (fifo_full) is still zero, and PC does not increment Cycle 6: 0x0, since dequeue_i, and thus sel_mux[1], is asserted only after the address makes it to SRAM Cycle 7: 0x1, since sel_mux[1] is asserted and the intruction at 0x0 has a p bit of 0, PC = PC + 1

6. Implementation of fetch unit in verilog

7. Resource utilization numbers are irrelevant to us for the time being since we have yet to implement controls to avoid having some of our resources remain idle. Once we implement these controls and pipelining resource utilization will be relevant. Additionally, since our partial processor does not contain any pipelining or multi-cycle funtionality, frequency numbers are essentially irrelevant as well.

8. We had to fix just about all of them.