Lab 5: Conditional Instruction Execution & Control Hazards

Objective: Implementation of a datapath designed for handling control-flow instruction execution in a RISC-V CPU.

Control-Flow Execution

In Lab 4 we created a preliminary version of our RISC-V CPU which **only supported sequential instruction execution.** Practically, control-flow execution is a significant part of a program. Branch type instructions allow decisions to be made by a program, such as if-else conditions and loops, which require the PC to be re-directed to fetch the correct instruction on the next cycle. In this lab, we will implement datapath circuitry to support control-flow. Lab 4 will be amended to support the branch instructions Branch-if-Equal (beq), Branch if not Equal (bne) and Branch-if-Less Than (blt) as shown in Fig. 1. The original IF stage is provided in the shaded grey box.

BEQ / BNE

The beq instruction has three operands, two registers that are compared for equality, and a 12-bit offset used to compute the branch target address, added to the PC. To implement this instruction, we must compute the branch target address by adding the sign-extended offset field of the instruction with the PC. The ISA also requires that the offset field be left shifted by 1 bit so that it is a half word offset; this shift increases the effective range of the offset field by a factor of 2. However, since we are working with limited memory, we can avoid this shift and simply add the two values. Note that it is possible to branch/jump back to a previous instruction by taking advantage of the signed offset.

To satisfy the BEQ instruction, two conditions must be met: 1) the control unit must assert a branch signal, and 2) the ALU must generate a zero flag after subtracting its two input operands. As seen in Fig. 1, if both conditions are met, the immediate value (PC) of the instruction may be forwarded from the "Imm Gen" unit, to the PC which loads the address. Thereafter, the BEQ flag will select the appropriate mux input. Recall the signals required to load the branch address into the PC unit from Lab 1. For BNE, if the zero flag is not asserted, the branch is taken (load PC with the immediate address).

BLT

BLT is similar to BEQ. The two conditions for BLT are 1) control unit must assert a branch, and 2) the ALU generates a carry flag after subtraction considering 2s complement, indicating that its second input is greater than first input operand. If both conditions are true, the PC obtains the "Imm Gen" unit value as the next address to fetch from Instruction memory (similar to BEQ).

Dealing with Control-Flow in the Pipeline

As the branch instruction's result (zero or carry flag) will only be known at the Exe cycle of the pipeline, the instruction fetch stage must stall until the outcome of the branch is know (referred to as "resolving" the branch). In the case of our RISC-V pipeline, it takes one clock cycle after Instruction Fetch (IF)/Decode to read/execute. Therefore, **whenever a branch type instruction is decoded, the IF unit must stall one (to two) cycles** to resolve the branch. Design the circuit required to stall upon a branch, and integrate the logic into your datapath design.

Testing your Control-Flow Datapath

Use BEQ, BLT and BNE to implement the following algorithm. Use this algorithm to test your datapath implementation for correctness:

```
int a = 2; int b = 3;
for(int i = 0; i < 3; i++){
     if(a ==b)
         a--;
     else
        b--;
}
```

You will need to convert the C code provided above to RISC-V assembly. Consult with the RISC-V instruction encoding in the Hennessy and Patterson textbook. Place the associated microcode in your instruction memory .mif, and any data values required for loading in your data memory .mif.

Data Hazards – as the case with the previous lab, avoid data hazards by using a variety of registers and at least two to three cycles between dependent instructions.

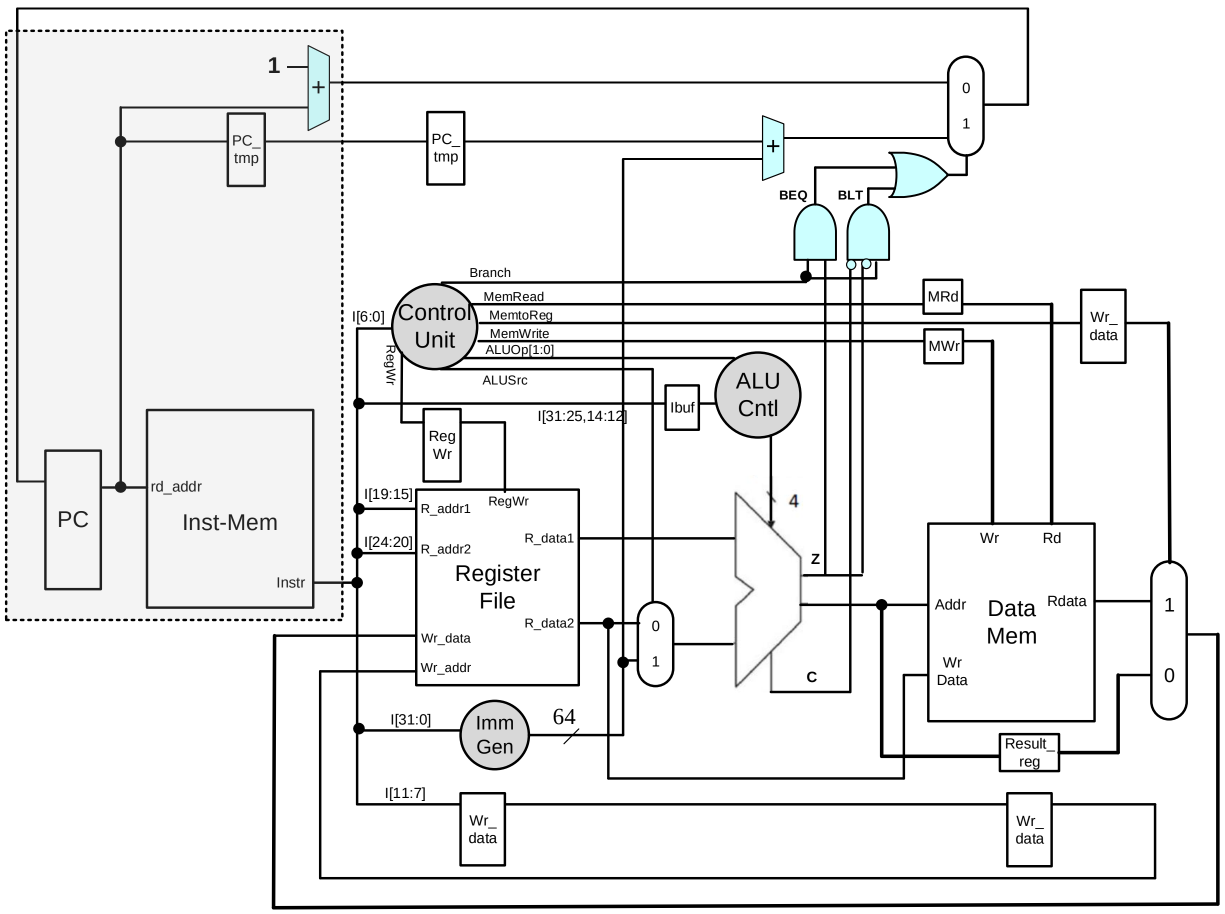

Fig.1: Datapath with Control-Flow Implementation (BNE not shown)

Deliverables:

- **VHDL** –all files necessary for datapath implementation and control-flow support
- **Mif** for instruction memory and data memory values
- **Timing** waveforms for the datapath design, implementing the specified algorithm
  - Recommended - create .wvf in Quartus and select **timing** simulation

```
Ld r1 a
Ld r2 b
Ld r13 #3 (counter)
Again :Sub r6, r1, r2
           BNE loop
           SUB r1, r1 #1
           SUB r20, r0, r0
           BEQ check
Loop:  SUB r2, r2, #1
Check: ADD r13, r13, #1
            SUB r17, r13, #2 (BGE)
            BGT again
```