Project #1

B05902120 / Yu-Ting, TSENG

Dec 16, 2018

Basic Execution

- Platform: linux (CSIE workstation)
-

1. The file related to `Register` and `Memory` has been modified in kinds of small degree.
2. The most important is to input clock to each memory between the stage which is the most critical point of the pipeline cycle.

```
always @(posedge clk_i) begin
    assign ...  // TODO
end
```

3. Some parts is as same as what we done during last assignment, the worth mentioning difference is "control siganl" and "mux".
4. We need a `control signal` to tell them what to do, and therefore we give each instruction a number.

```
assign  data_o = (data_i == {17'b00000001100110011}) ?
                 {4'b0000}:  // or
                 (data_i == {17'b00000001110110011}) ?
                 {4'b0001}:  // and
```

5. In order to decide which resource to use while calculating PC and arithmetric function, we add plenty of `mux` in the pipeline.

```
// mux of program counter
assign  data_o = (select_i == 0) ? data1_i:  // nextPC
                 (select_i == 1) ? data2_i:  // samePC
                 (select_i == 2) ? data3_i:  // branch
                 0;
```

```
// mux of doing instruction or flush
assign  control_o   = (select_i) ? 4'b1111:   // fluah
                      control_i;              // other

// mux for ALU source 1
assign  data_o = (control_i == 4'b0110) ? data1_i:
                 (control_i == 4'b0111) ? data1_i:
                 // store and load (reg1 addr)
                 (select_i == 2'b00) ? data2_i:
                 (select_i == 2'b01) ? data3_i:
                 (select_i == 2'b10) ? data4_i:
                 // other (reg1 and forwarding data)
                 0;

// mux for ALU source 2
assign  data_o = (control_i == 4'b0110) ? data1_i:
                 (control_i == 4'b0111) ? data1_i:
                 (control_i == 4'b0101) ? data1_i:
                 // store, load and I format(imm data)
                 (select_i == 2'b00) ? data2_i:
                 (select_i == 2'b01) ? data3_i:
                 (select_i == 2'b10) ? data4_i:
                 // other (reg2 and forwarding data)
                 0;
```

6. `sign extend` has a little bit change from last version, we choose the different input fopr different instructions.

```
assign  which   = (control_i == 4'b0111) ?
                  {data_i[31:25], data_i[11:7]}:
                  (control_i == 4'b1000) ?
                  {data_i[31], data_i[7], data_i
                     [30:25], data_i[11:8]}:
                  data_i[31:20];
                  // load, store, others in order
assign  data_o  = { {20{which[11]}}, which};
```

7. The most indispensable unit in this pipeline cycle is ALU. ALU can do plenty of things, such as calculating the address for the instruction load and store, or calculate the result for arithmetic instruction and choose the final consequence.

```
assign  cal_or = data1_i | data2_i;
assign  cal_and = data1_i & data2_i;
assign  cal_add = data1_i + data2_i;
assign  cal_sub = data1_i - data2_i;
assign  cal_mul = data1_i * data2_i;
                  // in order do operations
```

8. The unique part in the pipeline must be forward and hazard unit.

```
always@ (*) begin
// Hazard Detect
if (IDEX_MemRead &&
   ((IDEX_RDaddr_i == IFID_RSaddr1_i) ||
    (IDEX_RDaddr_i == IFID_RSaddr2_i)))
     hazard = 2'b1;              // hazard
else
    hazard = 2'b0;               // no hazard
end

always@ (*) begin
    // Forward A
    if (EXMEM_RegWrite && (EXMEM_RDaddr_i != 0) &&
       (EXMEM_RDaddr_i == IDEX_RSaddr1_i))
        ForwardA = 2'b10;        // EX forward of reg1
    else if (MEMWB_RegWrite && (MEMWB_RDaddr_i != 0)
       &&
            (MEMWB_RDaddr_i == IDEX_RSaddr1_i))
             ForwardA = 2'b01;   // MEM forward of reg1
    else ForwardA = 2'b00;       // no forward

    // Forward B
    if (EXMEM_RegWrite && (EXMEM_RDaddr_i != 0) &&
       (EXMEM_RDaddr_i == IDEX_RSaddr2_i))
```

```
            ForwardB = 2'b10;        // EX forward of reg2
        else if (MEMWB_RegWrite && (MEMWB_RDaddr_i != 0)
            &&
                (MEMWB_RDaddr_i == IDEX_RSaddr2_i))
                 ForwardB = 2'b01;   // MEM forward of reg2
        else ForwardB = 2'b00;       // no forward
    end
```