Computer Architecture Project1 Report

Members amd Team Work

> Link the sub-modules of the CPU
> Design some basic modules
> — B03902007 鄭德馨
>
> Handle stall and flush
> Report
> — B03902065 陳奕先
>
> Control the forwardings, including EX/MEM hazards and **forwarding to ID stage**
> Overall debugging and design the testbenches
> — B03902015 簡瑋德

Implementation of the Pipelined CPU

1. According to the datapath, determine needed components and modules
2. Design hazard detection and solutions, including forwarding, stall and flush
   - EX/MEM-Hazard: `MUX6`, `MUX7` and `FW`
   - **Forwarding to ID stage**: `MUX9`, `MUX10` and `FW`
   - Stall: `MUX8` and `HD`
   - Flush: `HD` and the registers in IF/ID
3. Determine all the control signal
4. Finish and combine all the above components

Implementation of Each Module

- `ALU` : Compute the output according to operation specified by ALUCtrl.
- `ALU_Control` : Assign value to ALUCtrl according to ALUOp and funct.
  - 000: add
  - 001: sub
  - 010: mul
  - 011: add
  - 100: or
- `Adder` : Add two input. Used to compute next PC address.
- `CPU` : Assign initial value to pipeline registers and connect ports of all modules. At each clock rising edge, update pipeline registers and do stall or flush if needed.

```
if (flush == 2'b01) // flush
begin
    IF_ID_pc        <= 32'd0;
    IF_ID_inst      <= 32'd0;
end
else if (flush == 2'b10) // read new
begin
    IF_ID_pc        <= added_inst_addr;
    IF_ID_inst      <= inst;
end
```

- `Control` : According to opcode, determine the instruction type. Then, set jump/branch signal if the instruction is j/beq. All other control signal are specified by the last 8 bits of the output.

    - control_o[0]: RegWrite
    - control_o[1]: MemToReg
    - control_o[2]: MemWrite
    - control_o[3]: MemRead
    - control_o[4]: RegDst
    - control_o[6:5]: ALUOp
    - control_o[7]: ALUSrc
- `Data_Memory` : If MemWrite is set, write to the memory. If MemRead is set, assign the data at corresponding memory to the output wire, otherwise assign zero to the output wire.
- `Equal` : When the two input are equal to each other, set the output to one, otherwise zero. Used when the instruction is beq.
- `FW` : Check whether there is EX hazard or MEM hazard and set forwarding control signal. Additionally, we implement mux9 and mux10 for forwarding to ID stage.

```
if(M_WB_RegWrite_i && M_WB_rd_i != 5'd0 && M_WB_rd_i == IF_ID_rs_i)
    mux9select_o <= 1'b1; // Forwarding to ID
else
    mux9select_o <= 1'b0;
if(M_WB_RegWrite_i && M_WB_rd_i != 5'd0 && M_WB_rd_i == IF_ID_rt_i)
    mux10select_o <= 1'b1; // Forwarding to ID
else
    mux10select_o <= 1'b0;
if(EX_M_RegWrite_i && EX_M_rd_i != 5'd0 && EX_M_rd_i == ID_EX_rs_i)
    mux6select_o <= 2'b10; // EX-Hazard
else if(M_WB_RegWrite_i && M_WB_rd_i != 5'd0 && M_WB_rd_i == ID_EX_rs_i)
    mux6select_o <= 2'b01; // MEM-Hazard
else
    mux6select_o <= 2'b00;
if(EX_M_RegWrite_i && EX_M_rd_i != 5'd0 && EX_M_rd_i == ID_EX_rt_i)
    mux7select_o <= 2'b10; // EX-Hazard
else if(M_WB_RegWrite_i && M_WB_rd_i != 5'd0 && M_WB_rd_i == ID_EX_rt_i)
    mux7select_o <= 2'b01; // MEM-Hazard
else
    mux7select_o <= 2'b00;
```

- `HD` : Detect if there is hazard, then set stall and flush control signal.

```
if(
    (MemRead_i == 1'b1 || (c_branch_i == 1'b1 && RegWrite_i == 1'b1) )
    && ( (ID_EX_rt_i == IF_ID_rs_i) || (ID_EX_rt_i == IF_ID_rt_i) )
)
begin
    stall_o <= 1'b1; // stall
    flush_o <= 2'b00; // keep IF/ID
    pc_hazard_o <= 1'b1; // PC do not change
end
else if (jump_i || branch_i)
begin
    stall_o <= 1'b1; // stall
    flush_o <= 2'b01; // flush IF/ID
    pc_hazard_o <= 1'b0; // change PC
end
else
begin
    stall_o <= 1'b0; // do not stall
    flush_o <= 2'b10; // do not flush
    pc_hazard_o <= 1'b0; // change PC
end
```

- `Instruction_Memory` : Assign instruction stored at the current PC address to the output.
- `MUX32` : If the select bit is 1/0, assign input data1/data2 to the output.
- `MUX32_3` : If the select bits are 00/01/others, assign input data1/data2/data3 to the output.
- `MUX5` : If the select bit is 1/0, assign input data1/data2 to the output. Different from MUX32, all data here are five bits.
- `PC` : If no hazard occurs, update PC address at each clock rising edge, otherwise keep current PC address and stall.
- `Registers` : Assign values stored at the certain registers to outputs. At each clock rising edge, write to corresponding register if RegWrite is set.
- `Shift_Left28` : Shift the input 2 bits to left. Used to compute the target address after jump instruction.
- `Shift_Left32` : Shift the input 2 bits to left. Used to compute the target address after beq instruction.
- `Sign_Extend` : Extend the input to 32 bits by appending the sign bit of input to its most significant side

Problems and Solutions

- Problem: Hard to Debug, since there’re so many ports and modules
- Solution: Besides unit-tests, we print the values of the ports and pipeline registers by modifying the testbench

Testbench Details

- Line 46: `CPU.Data_Memory.memory[0] = 8'h5; // n = 5 for example`
- Line 60: `if(counter == 70) // stop after 70 cycles`
- Line 63 and 64:

```
1  if (
2      CPU.HD.stall_o == 1'b1
3      && CPU.Control.jump_o == 1'b0
4      && CPU.Control.branch_o == 1'b0
5  ) stall = stall + 1;
6  if (CPU.HD.flush_o == 2'b01) flush = flush + 1;
```