Computer Architecture Project 1

Pipeline CPU Implementation

R E P O R T

B03902001 資工三 駱定暄
B03902027 資工三 王冠鈞
B03902039 資工三 施秉志

- Coding Enviroment
    1. System: 217 Workstation (Linux)
    2. Editor: Vim
    3. Collaborate via Github

- Module Implementation Details
    1. Adder (Adder.v): Trivial. Add two inputs.
    2. AND/OR (AND.v, OR.v): Trivial. Do one-bit AND/OR operation.
    3. Hazard Detection Unit (HDU.v) : Check the instruction in ID/EX has the signal MemRead on. If so, then check whether there are any data deendicies, i.e. the register Rt in the ID/EX state is same as either Rt or Rs in the IF/ID state. If so, output the stall signal to the modules PC, IF/ID latch, and Mux8. The following instructions may trigger it to send a stall signal:
       lw $t1, 0($s0)   // MemRead = 1; Rt = $t1
       add $t2, $t0, $t1 // Rt = $t1
    4. Multiplexers (MUX5.v, MUX8.v, MUX32.v, MUX32_3.v): Trivial. Different multiplexers differ from I/O data size, and the number of input data.
    5. Pipeline Latch (IFID.v, IDEX.v, EXMEM.v, MEMWB.v): New values are written into the registers in pipeline latches at rising edge. In addition, latch IFID also have to handle stall and flush. When a stall (Stall_i) occurs, the output registers keep their old value. When the flush signal (Flush_i) is on, the output instruction and the program counter will be assigned to zero.
    6. Equality Test (EQ.v): Compare two 32-bit values. If two values are the same, it will send out 1’b1 as a signal.
    7. PC (PC.v): Provided by TAs, so we only add stall handling. The output program counter will keep the old value, if the stall signal(stall_i) is on.

8. Registers (Registers.v): Provided by TAs, so the only change is that the time for writing into register is now at falling edge to solve ID forwarding.
9. Sign Extension (Sign_Extend.v): Extend a 16-bit data(data_i) to a 32-bit data(data_o):
   data_o = {{16{data_i[15]}}, data_i}
10. Shift Left (Sll.v): Shift the input data(data_i) to the left by the amount of a 4-bit value(lshift).
11. Count Stall and Flush (testbench.v):
    Stall:
    if (CPU.HDU.mux8_o == 1 &&
    CPU.Control.Jump_o == 0 &&
    CPU.Control.Branch_o == 0) stall = stall + 1;
    Flush:
    if (CPU.OR_Flush.or_o == 1) flush = flush + 1;
12. Forwarding Unit (FWD.v):
    if (1)RegWrite signal(EXMEM_RegWr_i) is on at stage EX/MEM , (2)the address of rd(EXMEM_RegRd_i) at MEM stage is not $zero, and (3)the addresses of input data(IDEX_RegRs_i or IDEX_RegRt_i) equal to the address of register store destination(EXMEM_RegRd_i) at MEM stage:
        forward the data at MEM stage to EX stage
    else if (1)RegWrite signal(MEMWB_RegWr_i) is on at stage MEM/WB , (2)the address of rd(MEMWB_RegRd_i) at WB stage is not $zero, and (3)the addresses of input data(IDEX_RegRs_i or IDEX_RegRt_i) equal to the address of register store destination(MEMWB_RegRd_i) at WB stage:
        forward the data at WB stage to EX stage
    else:
        do normal execution
    endif

13. Data Memory (Data_Memory.v):
    if MemWrite(Data_Memory.MemWrite_i) signal is on:
        write four bytes of data from input (Data_Memory.data_i) into data memory starting from the given address (Data_Memory.addr_i).
    endif
    if MemRead(Data_Memory.MemRead_i) signal is on:
        read four bytes of data starting from the given address (Data_Memory.addr_i) and send it to the output (Data_Memory.data_o)

endif

14. CPU (CPU.v): Connection between each module is done by wires. The data flow and control flow follow the architecture shown on the last page of the instruction file (project1.pdf) provided by TAs.
15. ALU Control Signal(ALU_Control.v): Decide what ALU should do according to ALUOp and function bits:

ALUOp\funct_i	ADD	SUB	AND	OR	MUL	default
R-type	3'b000 (add)	3'b001 (sub)	3'b010 (and)	3'b011 (or)	3'b100 (mul)	3'b111
other	3'b000 (addi, lw,sw, j,beq)	x	x	x	x	x

16. (Control.v): Decide the control signals with respect to the operation code:

	addi (001000)	R-type (000000)	lw (100011)	sw (101011)	beq (000100)	j (000010)	other (?)
RegDst	0	1	0	x	x	x	0
ALUSrc	1	0	1	1	1	0	0
MemtoReg	0	0	1	x	x	x	0
RegWrite	1	1	1	0	0	0	0
Memwrite	0	0	0	1	0	0	0
MemRead	0	0	1	0	0	0	0
Branch	0	0	0	0	1	0	0
Jump	0	0	0	0	0	1	0
ALUOp	2'b00	2'b10	2'b00	2'b00	2'bxx	2'bxx	2'b00

17. (ALU.v): Get control signals from ALU control unit (ALU_Control.v) and perform the wanted execution.

- Testing Method
    1. Simulator: iverilog
    2. Wave Viewer: gtkwave (for debug purpose)
    3. Testing(=constructing) Sequence:
        a. ‘lw/sw’
        b. ‘branch’
        c. ‘jump’
        d. pipeline
        e. fowarding
        f. hazard detecting

    We implemented these parts step by step to do unit testing.
    Some testbenches were created to test the part we added in every step.

- Task Distribution

    駱定暄：new instructions implementation, pipeline
    王冠鈞：hazard detection (stall and flush)
    施秉志：forwarding