Lab06 实验报告

王正 518021910079

一、 实验名称

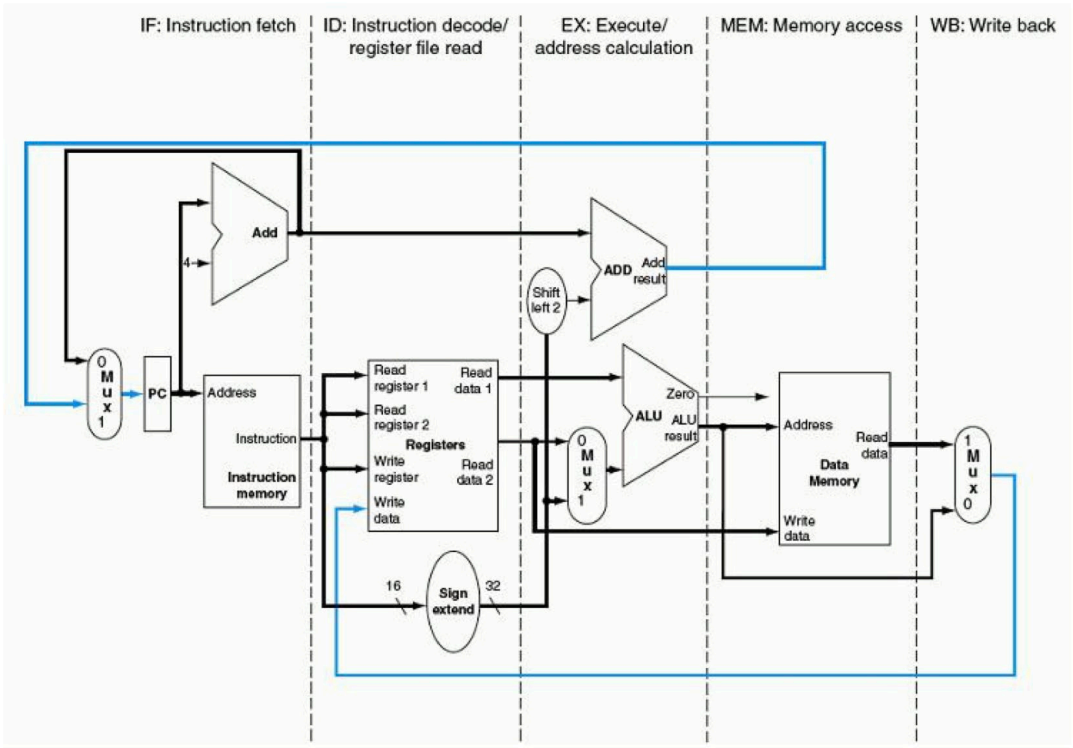
FPGA 基础实验: 简单的类 MIPS 多周期流水线处理器设计与实现

二、 实验目的

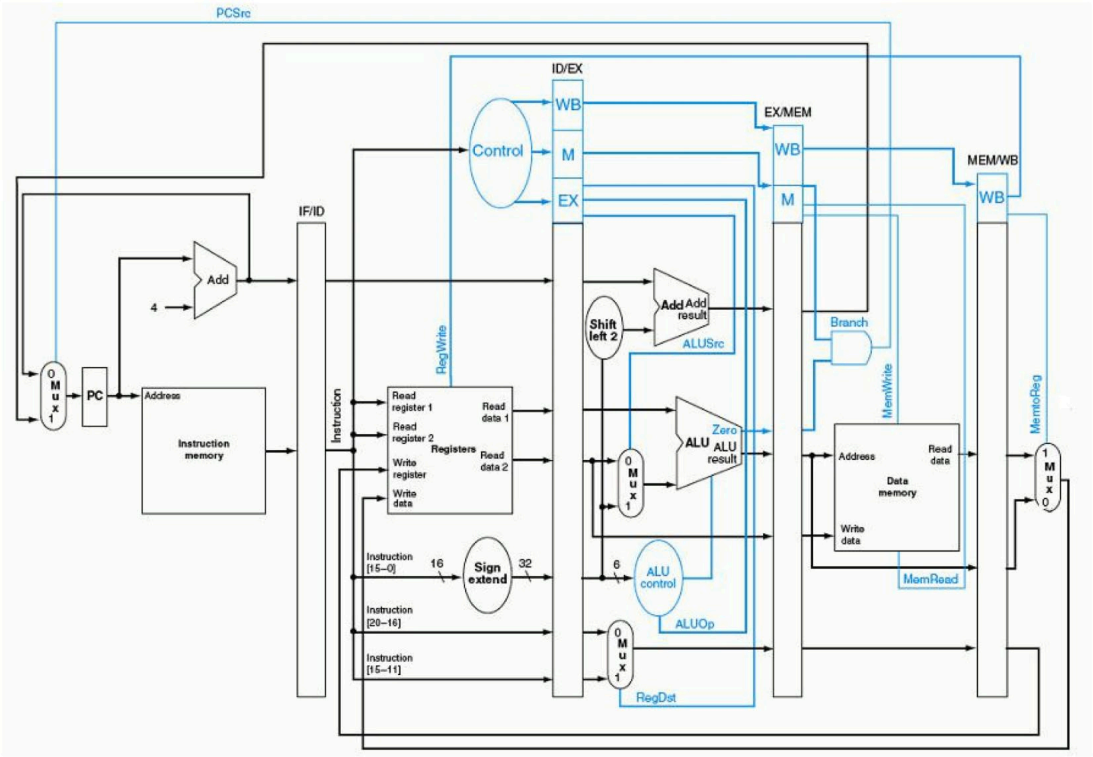
1. 理解 CPU 的流水线，了解流水线的相关（dependence）和冒险（hazard）；
2. 设计支持停顿（stall）的流水线 CPU，，通过检测竞争并插入停顿机制解决数据冒险、控制冒险和结构冒险;
3. 增加转发（forwarding）机制解决数据竞争，减少因数据竞争带来的流水线停顿延时，提高流水线处理器性能;
4. 通过 predict-not-taken 或延时转移策略解决控制冒险/竞争，减少控制冒险带来的流水线停顿;

三、 实验原理

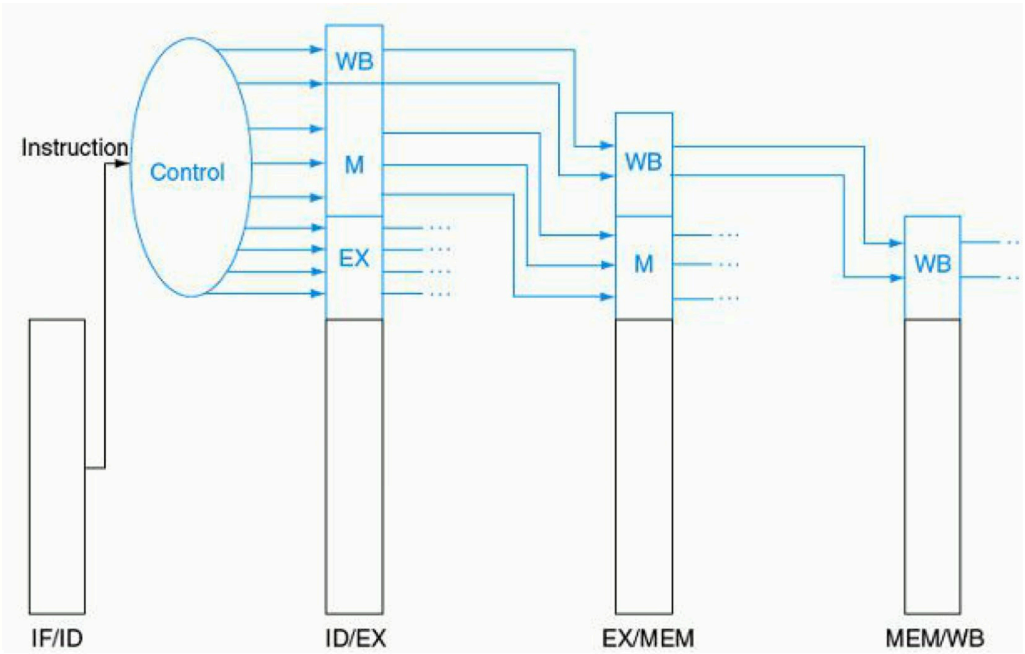
根据流水的主要结构:



将单周期 CPU 进行分割，插入 4 级寄存器，划分为 IF,ID,EX,MEM,WB 五个部分:



其中 Control 的输出需要被加入流水线寄存器保存下来，令后续每级流水线使用:



四、 功能实现

大部分功能模块与 Lab05 相同，在此不再赘述。着重介绍本次实验的顶层模块 Top。

1. 4 级流水线寄存器

```
// IF and ID
reg [31:0] IFID_pcPlus4, IFID_inst;
wire [4:0] IFID_INST_RS = IFID_inst[25:21];
wire [4:0] IFID_INST_RT = IFID_inst[20:16];
wire [4:0] IFID_INST_RD = IFID_inst[15:11];

// ID and EX
reg [31:0] IDEX_readData1;
reg [31:0] IDEX_readData2;
reg [31:0] IDEX_IMMSext;
reg [4:0] IDEX_inst_Rs;
reg [4:0] IDEX_inst_Rt;
reg [4:0] IDEX_inst_Rd;
reg [8:0] IDEX_Ctr;

wire IDEX_REGDST = IDEX_Ctr[8];
wire [1:0] IDEX_ALUOP = IDEX_Ctr[7:6];
wire IDEX_ALUSRC= IDEX_Ctr[5];
wire IDEX_BRANCH = IDEX_Ctr[4];
wire IDEX_MEMREAD = IDEX_Ctr[3];
wire IDEX_MEMWRITE = IDEX_Ctr[2];
wire IDEX_REGWRITE = IDEX_Ctr[1];
wire IDEX_MEMTOREG = IDEX_Ctr[0];


// EX and EM
reg [31:0] EXMEM_aluRes, EXMEM_writeData;
reg [4:0] EXMEM_dstReg;
reg [4:0] EXMEM_Ctr;
reg EXMEM_zero;
wire EXMEM_BRANCH = EXMEM_Ctr[4];
wire EXMEM_MEMREAD = EXMEM_Ctr[3];
wire EXMEM_MEMWRITE = EXMEM_Ctr[2];
wire EXMEM_REGWRITE = EXMEM_Ctr[1];
wire EXMEM_MEMTOREG = EXMEM_Ctr[0];


// MEM and WB
reg [31:0] MEMWB_readData, MEMWB_aluRes;
reg [4:0] MEMWB_dstReg;
reg [1:0] MEMWB_Ctr;
wire MEMWB_REGWRITE = MEMWB_Ctr[1];
wire MEMWB_MEMTOREG = MEMWB_Ctr[0];
```

2. 五阶段实现

在流水线中，有多条指令在不同的阶段并行执行，为了避免产生冲突，需要对存储器和寄存器的读写进行同步。因此设置时钟下降沿对寄存器和数据存储器进行写入，上升沿对流水线寄存器进行写入。这样对于寄存器和数据存储器，在前半个周期写入，后半个周期读取，能够解决一部分结构冒险和数据冒险。

- IF 取指

  PC 在每个时钟上升沿更新。当发生插入停顿时（STALL）PC 停止更新。当出现 BRANCH 时，要将已取出的指令清空。

```
// IF
wire [31:0] IF_INST;
wire [31:0] PC_PLUS_4;
wire [31:0]  BRANCH_ADDR;
wire [31:0] next;

assign PC_PLUS_4 = PC + 4;

mux32 nextMux(
    .input2(PC_PLUS_4),
    .input1(BRANCH_ADDR),
    .out(next),
    .sel(BRANCH)
);


inst_memory instMem(
    .readAddress(PC),
    .inst(IF_INST)
);

always @ (posedge clk)
begin
    if (!STALL)
    begin
        IFID_pcPlus4 <= PC_PLUS_4;
        IFID_inst <= IF_INST;
        PC <= next;
    end
    if (BRANCH)
        IFID_inst <= 0;
end
```

- ID 译码

该部分包含了跳转的判定和跳转地址的计算，这样可以使跳转更早发生，可以减少清楚流水线寄存器的次数，简化流水线。当插入停顿发生时，需将所有控制信号置零。

```
// ID


wire [31:0] IMM_SEXT_SHIFT = IMM_SEXT << 2;

assign BRANCH_ADDR = IMM_SEXT_SHIFT + IFID_pcPlus4;
assign BRANCH = (READ_DATA_1 == READ_DATA_2) & CTRL_OUT[4];

always @ (posedge clk)
begin
    IDEX_Ctr <= STALL ? 0 : CTRL_OUT;
    IDEX_readData1 <= READ_DATA_1;
    IDEX_readData2 <= READ_DATA_2;
    IDEX_IMMSext <= IMM_SEXT;
    IDEX_inst_Rs <= IFID_INST_RS;
    IDEX_inst_Rt <= IFID_INST_RT;
    IDEX_inst_Rd <= IFID_INST_RD;
end
```

- EX 执行

与 Lab05 的差别不大

```
// EX
wire [31:0] ALU_SRC_A = fEX_A ? EXMEM_aluRes : fMEM_A ? REG_WRITE_DATA : IDEX_readData1;
wire [31:0] ALU_SRC_B = IDEX_ALUSRC ? IDEX_IMMSext : fEX_B ? EXMEM_aluRes : fEX_B ? EXMEM_aluRes : fMEM_B ? REG_WRITE_DATA : IDEX_readData2;
wire [31:0] MEM_WRITE_DATA = fEX_B ? EXMEM_aluRes : fEX_B ? EXMEM_aluRes : fMEM_B ? REG_WRITE_DATA : IDEX_readData2;


ALUCtr aluCtr(
    .Funct(IDEX_IMMSext[5:0]),
    .ALUOp(IDEX_ALUOP),
    .ALUCtrOut(ALU_CTRL_OUT)
);

ALU alu(
    .input1(ALU_SRC_A),
    .input2(ALU_SRC_B),
    .aluCtr(ALU_CTRL_OUT),
    .zero(ZERO),
    .aluRes(ALU_RES)
);

mux5 DstRegMux(
    .sel(IDEX_REGDST),
    .input1(IDEX_inst_Rd),
    .input2(IDEX_inst_Rt),
    .out(DST_REG));

always @ (posedge clk)
begin
    EXMEM_Ctr <= IDEX_Ctr[4:0];
    EXMEM_zero <= ZERO;
    EXMEM_aluRes <= ALU_RES;
    EXMEM_writeData <= MEM_WRITE_DATA;
    EXMEM_dstReg <= DST_REG;
end
```

- MEM/WB 访存和写回

  与单周期处理器差别不大:

```
// MEM
wire [31:0] MEM_READ_DATA;

dataMemory dataMem(
    .Clk(clk),
    .address(EXMEM_aluRes),
    .writeData(EXMEM_writeData),
    .memRead(EXMEM_MEMREAD),
    .memWrite(EXMEM_MEMWRITE),
    .readData(MEM_READ_DATA)
);

always @ (posedge clk)
begin
    MEMWB_Ctr <= EXMEM_Ctr[1:0];
    MEMWB_readData <= MEM_READ_DATA;
    MEMWB_aluRes <= EXMEM_aluRes;
    MEMWB_dstReg <= EXMEM_dstReg;
end

 // WB
 mux32 writeDataMux(
     .sel(MEMWB_MEMTOREG),
     .input1(MEMWB_readData),
     .input2(MEMWB_aluRes),
     .out(REG_WRITE_DATA));
```

3. 转发和插入停顿

根据体系结构的理论知识，转发是指将之前周期的执行阶段或访存阶段得到的数据，直接作为当前执行阶段的操作数之一。通过查找一些资料，我找到了相应的逻辑表达式:

```
// Forward
wire fEX_A = EXMEM_REGWRITE & EXMEM_dstReg != 0 & EXMEM_dstReg == IDEX_inst_Rs;
wire fEX_B = EXMEM_REGWRITE & EXMEM_dstReg != 0 & EXMEM_dstReg == IDEX_inst_Rt;
wire fMEM_A =
    MEMWB_REGWRITE & MEMWB_dstReg != 0 &
    !(EXMEM_REGWRITE & EXMEM_dstReg != 0 & EXMEM_dstReg != IDEX_inst_Rs) &
    MEMWB_dstReg == IDEX_inst_Rs;
wire fMEM_B =
    MEMWB_REGWRITE & MEMWB_dstReg != 0 &
    !(EXMEM_REGWRITE & EXMEM_dstReg != 0 & EXMEM_dstReg != IDEX_inst_Rt) &
    MEMWB_dstReg == IDEX_inst_Rt;
```

由此可以得出 ALU 的两个源操作数的表达式:

```
wire [31:0] ALU_SRC_A = fEX_A ? EXMEM_aluRes : fMEM_A ? REG_WRITE_DATA : IDEX_readData1;
wire [31:0] ALU_SRC_B = IDEX_ALUSRC ? IDEX_IMMSext : fEX_B ? EXMEM_aluRes : fEX_B ? EXMEM_aluRes : fMEM_B ? REG_WRITE_DATA : IDEX_readData2;
wire [31:0] MEM_WRITE_DATA = fEX_B ? EXMEM_aluRes : fEX_B ? EXMEM_aluRes : fMEM_B ? REG_WRITE_DATA : IDEX_readData2;
```

当 lw 指令后的指令直接访问 lw 所加载的寄存器的数据，此时无法通过转发来解

决数据冒险，只能够暂停流水线，即 STALL。

```
// Hazard detection
wire STALL = IDEX_MEMREAD &
    (IDEX_inst_Rt == IFID_INST_RS | IDEX_inst_Rt == IFID_INST_RT);
```

五、 结果展示

- 测试程序

```
lw $4, 0($0)
loop:
lw $1, 0($0)
lw $2, 4($0)
add $3, $2, $0
or $1, $3, $1
add $5, $2, $4
sw $5, 16($0)
slt $6, $2, $5
beq $2, $3, loop
add $1, $1, $4
```

- 测试数据:

```
0x00-0x03 5
0x04-0x07 7
```

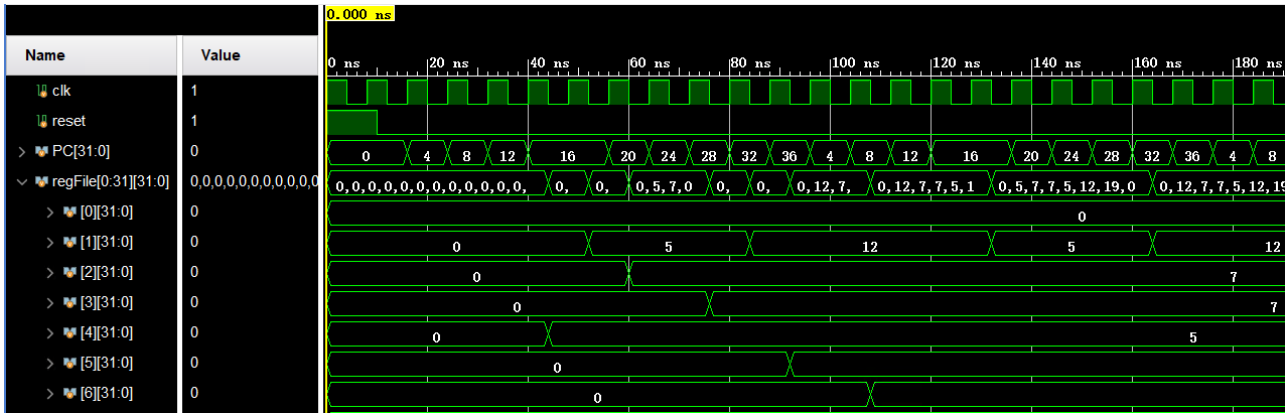
- 仿真激励文件

clk 周期为 8ns，reset 保持高电平 10ns:

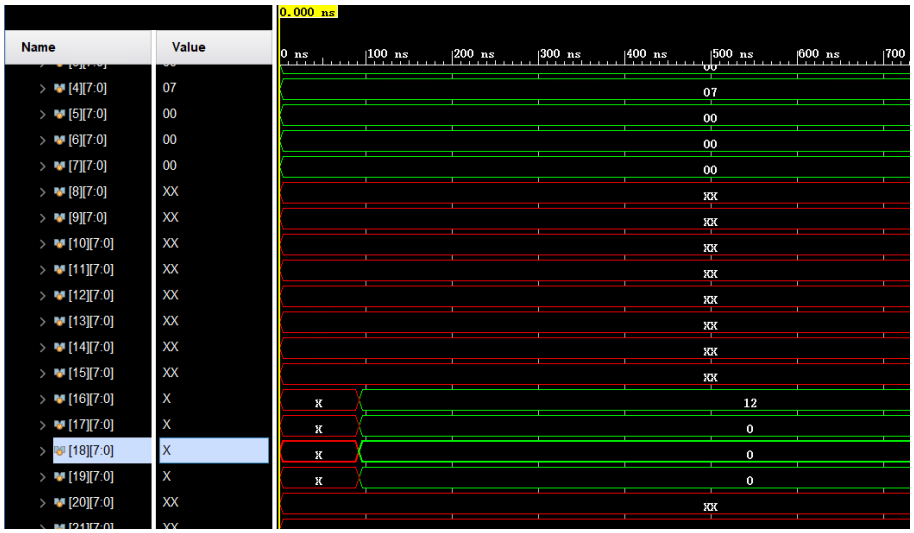
```
module Top_tb();

    reg clk;
    reg reset;

    always #4 clk=!clk;
    Top u0(
        .clk(clk),
        .reset(reset)
    );
    initial begin
        clk=1;
        reset=1;
        #10
        reset=0;
        #2000;
    end
endmodule
```

- 仿真波形

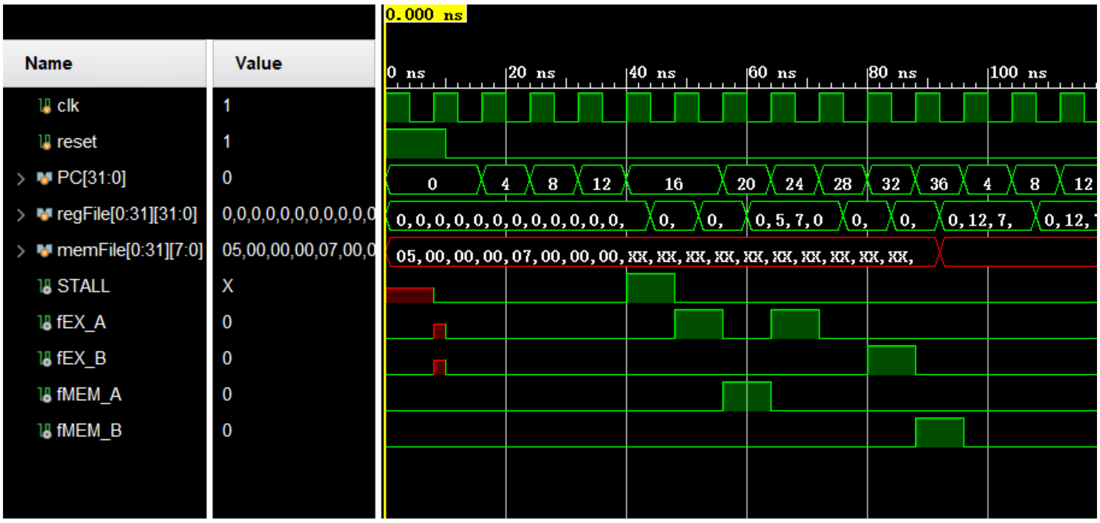
部分寄存器:



部分数据存储器:



停顿及转发信号:



在循环中，lw 到 add 出现了一次停顿;

add 到 or 及 add 到 sw 出现两次从 EX 阶段到第一个操作数的转发;

sw 到 slt 有一次从 EX 阶段到第二个操作数的转发;

lw 到 add 有一次从 MEM 到第一阶段操作数的转发;

add 到 slt 有一次从 MEM 到第二阶段操作数的转发;

六、 心得体会

本次实验是这次课程的最后一个实验，同时也是最难的一个实验。不仅需要充分理解流水线 CPU 的工作原理，还要正确排写大量的线路和寄存器。整个实验不仅耗时还需要有足够的耐心和细心。仿真出现错误时，需要根据线路图一点点地追溯相关数据和信号。

但是通过这次实验，我对流水线 CPU 的工作原理有了一个更加深入的认识。流水线 CPU 线路复杂，指令间相关性很强，用代码来实现理论课上学习到的抽象知识，有助我对体系结构的知识进行巩固。

很遗憾的是我没有时间去实现选做题的要求，希望以后有机会能有一个更深入的探索。

本学期的系统结构实验课到此结束了，这门课使我对系统结构的理解更上一层楼，难度逐渐增大的 6 个实验也让我学到了很多东西！最后要说一声：谢谢老师的指导！老师您辛苦了！