计算机系统结构实验

实验 6 报告

类MIPS多周期流水线处理器的设计与实现

李子龙

518070910095

2021 年 6 月 9 日

目录

1 实验目的

1. 理解CPU Pipeline，了解流水线冒险(hazard)及相关性，设计基础流水线CPU
2. 增加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
3. 设计支持Stall的流水线CPU。通过检测竞争并插入停顿（Stall）机制解决数据冒险、控制竞争和结构冒险
4. 通过predict-not-taken或延时转移策略解决控制冒险/竞争，减少控制竞争带来的流水线停顿延时，进一步提高处理器性能
5. 将CPU支持的指令数量从16条扩充为31条，使处理器功能更加丰富（选做，本次仅完成 16 条）

2 原理分析

2.1 基础流水线

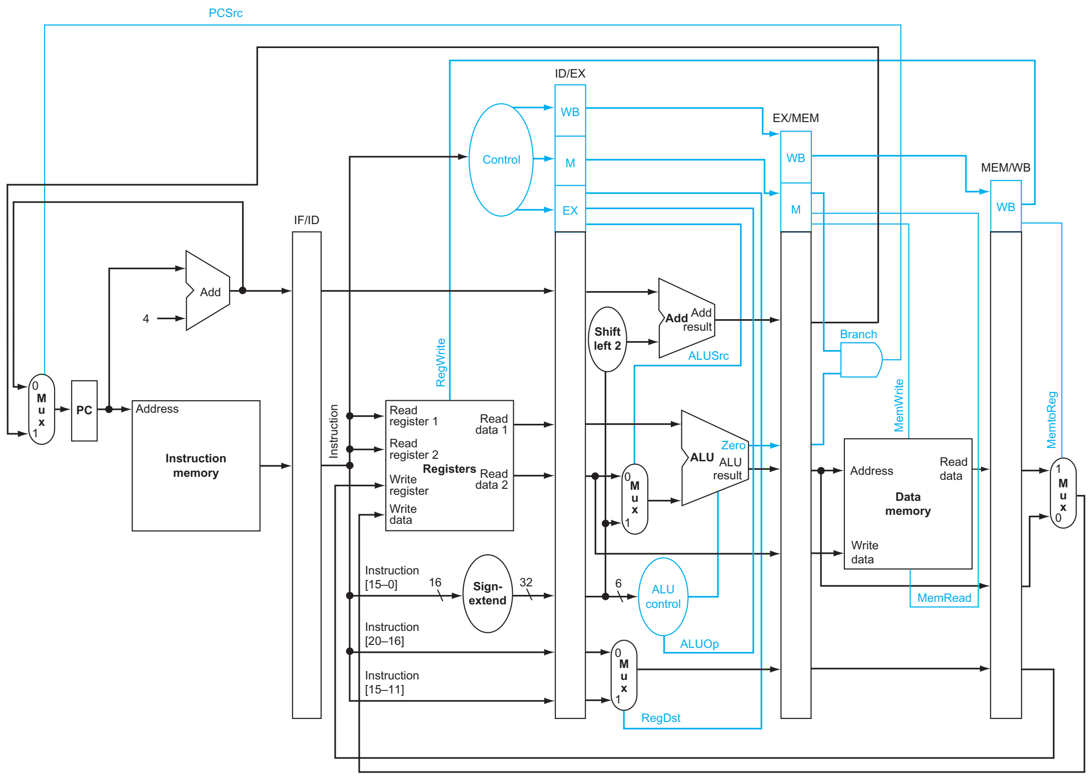

FIGURE 4.51 The pipelined datapath of Figure 4.46, with the control signals connected to the control portions of the pipeline registers. The control values for the last three stages are created during the instruction decode stage and then placed in the ID/EX pipeline register. The control lines for each pipe stage are used, and remaining control lines are then passed to the next pipeline stage.

2.2 前向转发机制

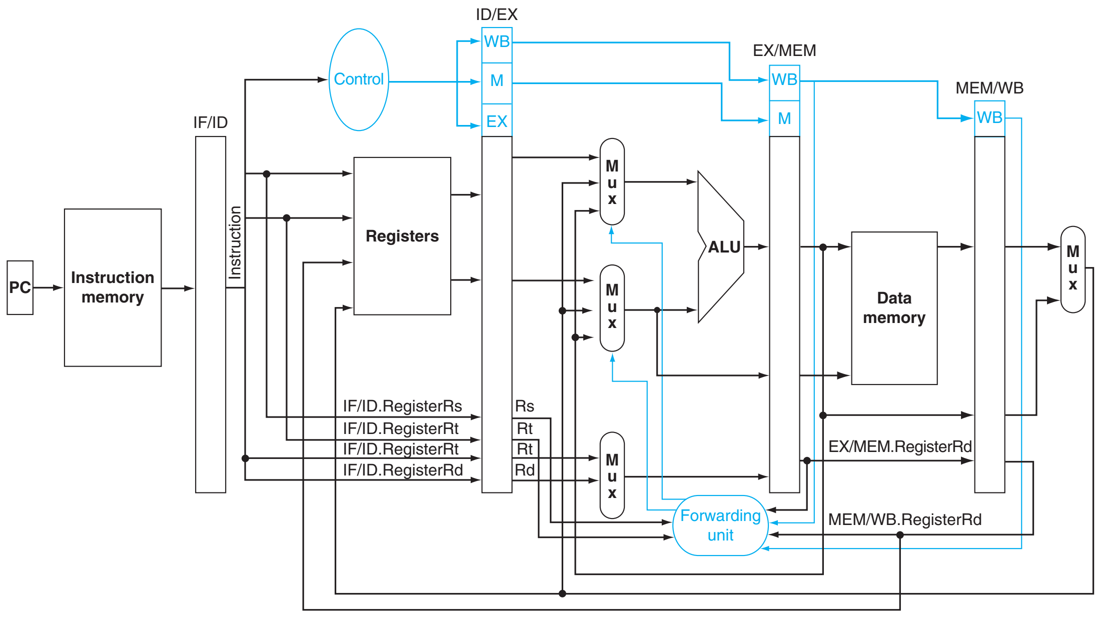

FIGURE 4.56 The datapath modified to resolve hazards via forwarding. Compared with the datapath in Figure 4.51, the additions are the multiplexors to the inputs to the ALU. This figure is a more stylized drawing, however, leaving out details from the full datapath, such as the branch hardware and the sign extension hardware.

多选器控制	源	解释
ForwardA=00	ID	第一个 ALU 操作数来自寄存器堆
ForwardA=10	EX	第一个 ALU 操作数由上一个 ALU 运算结果转发获得
ForwardA=01	MEM	第一个 ALU 操作数从数据存储器或者前面的 ALU 运算结果中转发获得
ForwardB=00	ID	第二个 ALU 操作数来自寄存器堆
ForwardB=10	EX	第二个 ALU 操作数由上一个 ALU 运算结果转发获得
ForwardB=01	MEM	第二个 ALU 操作数由数据存储器或者前面的 ALU 结果转发获得

EX 冒险:

```
if (EX_REG_WRITE &&
    (EX_WRITE_REG != 0) &&
    (EX_WRITE_REG == ID_INST[25:21]))
    ForwardA = 10;
if (EX_REG_WRITE &&
    (EX_WRITE_REG != 0) &&
    (EX_WRITE_REG == ID_INST[20:16]))
    ForwardB = 10;
```

MEM 冒险：

```
if (MEM_REG_WRITE && (MEM_WRITE_REG != 0)
    && !(EX_REG_WRITE && (EX_WRITE_REG != 0)
        && (EX_WRITE_REG != ID_INST[25:21]))
    && (MEM_WRITE_REG == ID_INST[25:21]))
    ForwardA = 01;
if (MEM_REG_WRITE && (MEM_WRITE_REG != 0)
    && !(EX_REG_WRITE && (EX_WRITE_REG != 0)
        && (EX_WRITE_REG != ID_INST[20:16]))
    && (MEM_WRITE_REG == ID_INST[20:16]))
    ForwardB = 01;
```

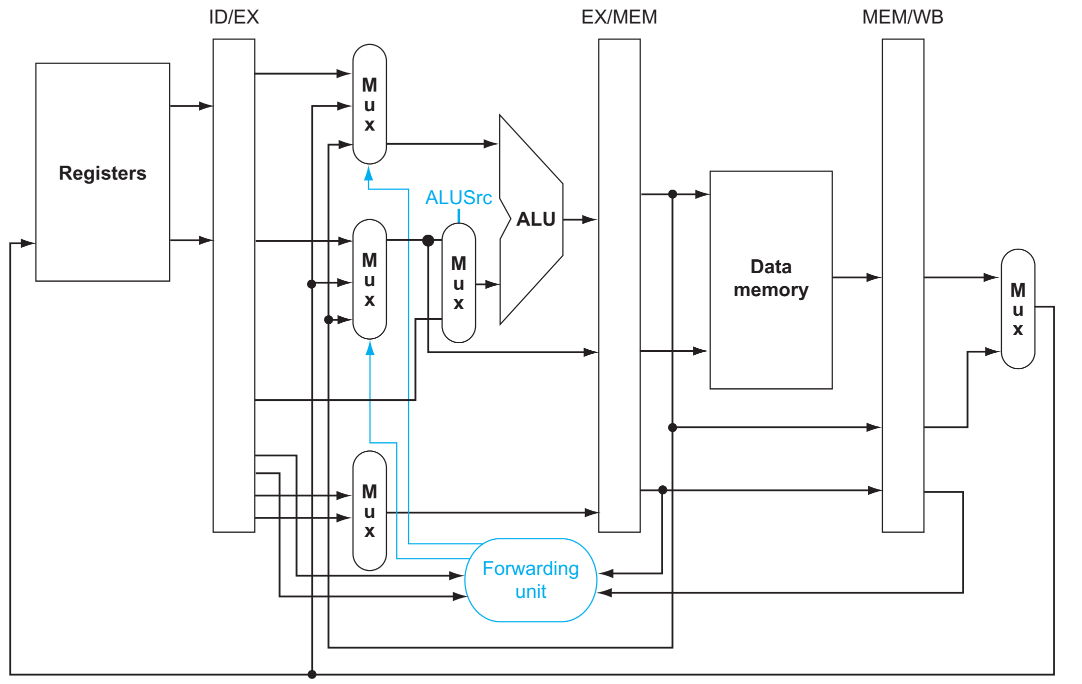

FIGURE 4.57 A close-up of the datapath in Figure 4.54 shows a 2:1 multiplexor, which has been added to select the signed immediate as an ALU input.

```
ALU alu(
    .input1(SHAMT ? ID_INST[10:6] :
        (ForwardA == 2'b00 ? ID_READ_DATA1 :
            (ForwardA == 2'b10 ? EX_ALU_RES :
                WRITE_DATA_WB))
    ),
    .input2(ID_ALU_SRC ? ID_OPAND :
        (ForwardB == 2'b00 ? ID_READ_DATA2 :
            (ForwardB == 2'b10 ? EX_ALU_RES :
                WRITE_DATA_WB))
    ),
    .aluCtr(ALU_CTR),
    .zero(ZERO_EX),
    .aluRes(ALU_RES_EX)
);
```

2.3 停顿机制

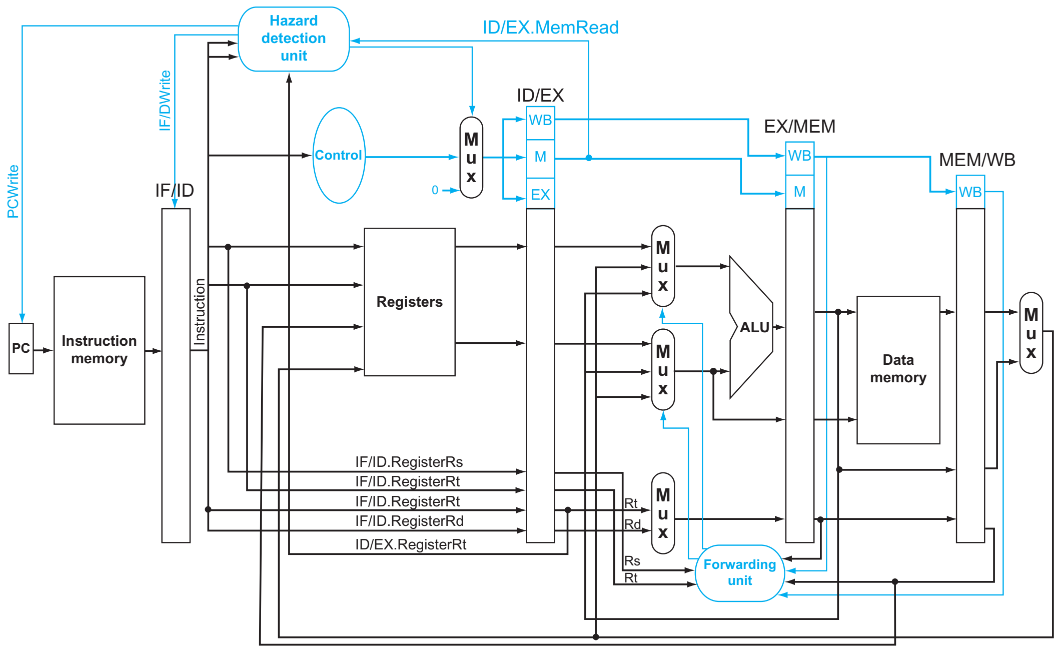

FIGURE 4.60 Pipelined control overview, showing the two multiplexors for forwarding, the hazard detection unit, and the forwarding unit. Although the ID and EX stages have been simplified—the sign-extended immediate and branch logic are missing—this drawing gives the essence of the forwarding hardware requirements.

```
wire stalling = (ID_MEM_READ &&
    ((ID_INST[20:16] == IF_INST[25:21])
    || (ID_INST[20:16] == IF_INST[20:16]))) ?
        1 : 0;
```

2.4 预测不发生机制

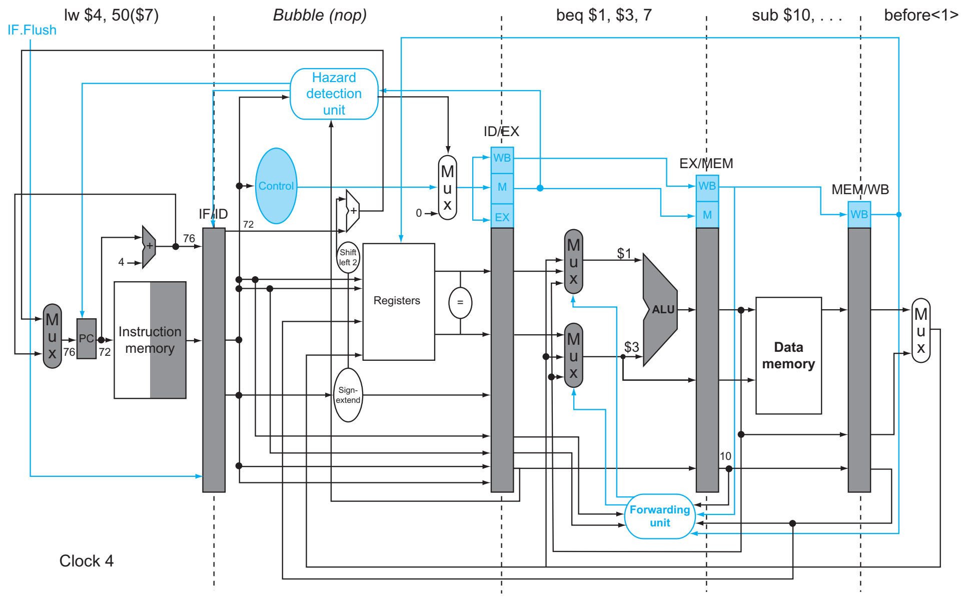

FIGURE 4.62 The ID stage of clock cycle 3 determines that a branch must be taken, so it selects 72 as the next PC address and zeros the instruction fetched for the next clock cycle. Clock cycle 4 shows the instruction at location 72 being fetched and the single bubble or nop instruction in the pipeline as a result of the taken branch. (Since the nop is really sll $0, $0, 0, it's arguable whether or not the ID stage in clock 4 should be highlighted.)

将 branch 判定提前到 ID 段。

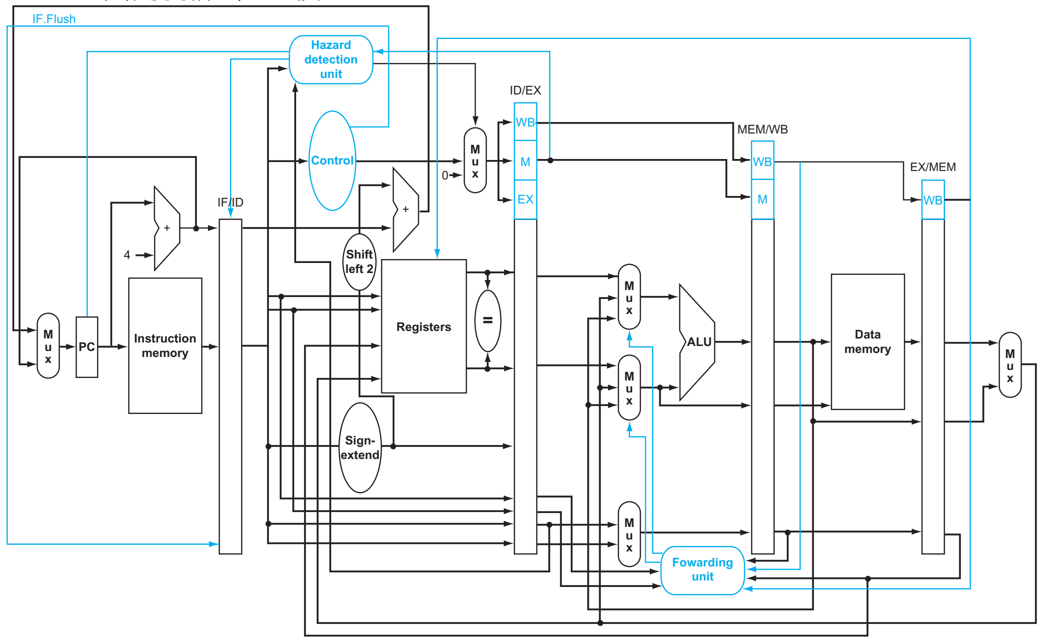

FIGURE 4.65 The final datapath and control for this chapter. Note that this is a stylized figure rather than a detailed datapath, so it's missing the ALUsrc Mux from Figure 4.57 and the multiplexor controls from Figure 4.51.

3 仿真结果

4 实验心得