上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

计算机系统结构实验报告

实验名称：简单的类 MIPS 多周期流水化处理器实现

姓　　名：洪瑄锐

学　　号：517030910227

班　　级：F1703302

手　　机：15248246044

邮　　箱：1204378645@qq.com

目录

1. 实验目的

1) 理解 CPU 的 Pipeline，了解流水线的相关和冒险（hazard）。

2) 设计流水线 CPU，支持 Stall。通过检测竞争并插入停顿（Stall）机制解决数据冒险、控制冒险和结构冒险。

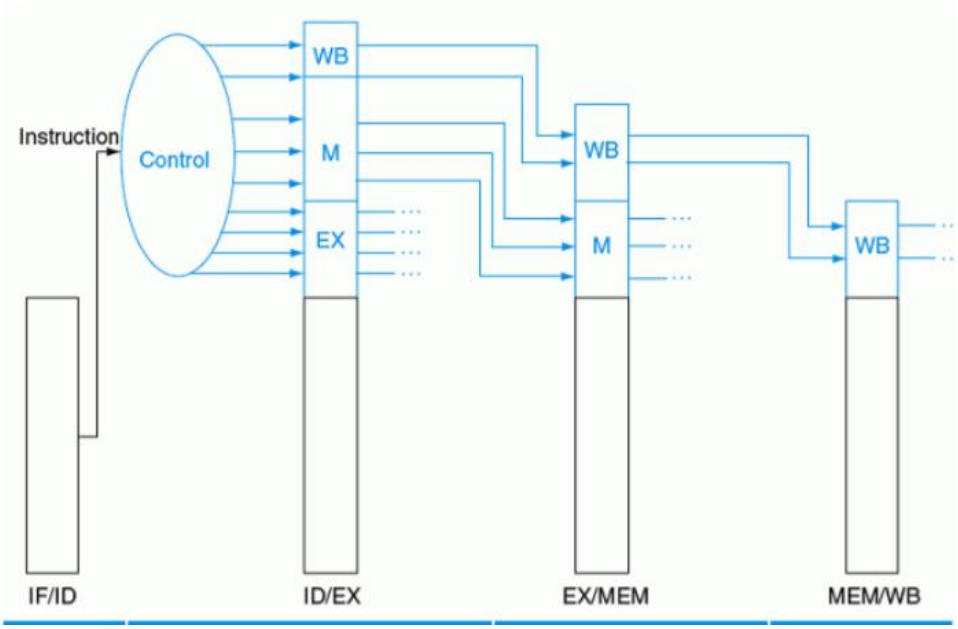
2. 实验原理与代码实现

2. 1 指令流水线的实现

2.1.1 主要思想

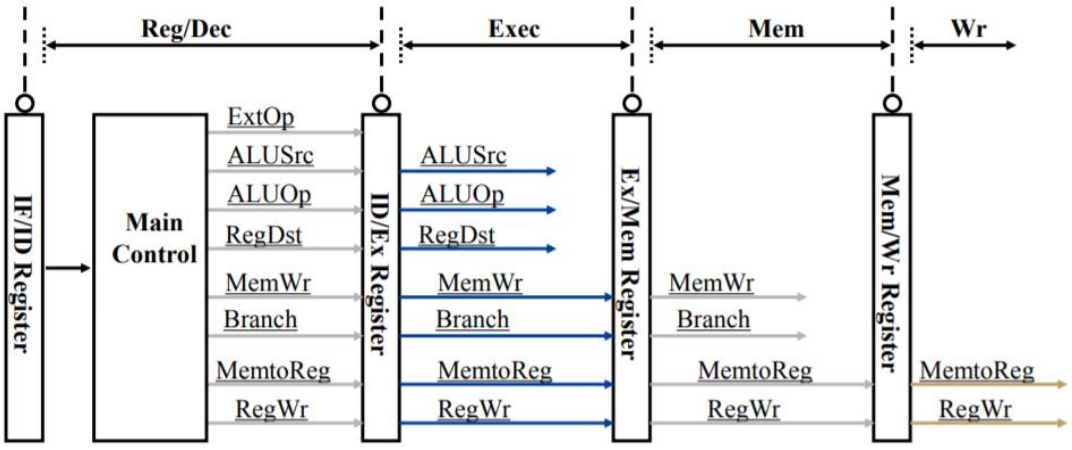
一条指令的执行分为取指（IF），译码（ID），执行（EX），访存（MEM），写回（WB）五个步骤，在单周期 CPU 中当一条指令执行完所有步骤下一条指令才开始取指，但对于多周期 CPU，一条指令在当前周期取指完毕进入下一步骤译码时，该指令的下一条指令立即开始取指，构成指令流水线，大大提高了 CPU 运行效率。

与单周期 CPU 实现不同的是，多周期 CPU 需要段寄存器传递控制信号，主控制单元在译码阶段产生所有的控制信号，EX 段需要的控制信号在一周期后使用，Mem 段需要的控制信号在两周期后使用，Wr 段需要的控制信号在三周期后使用。



如图所示，我们共需要四种段寄存器，连接 IF 与 ID 的段寄存器，连接 ID 与 EX 的段寄存器，连接 EX 与 MEM 的段寄存器，连接 MEM 与 WB 的段寄存器。其中 ID/EX 需要存储 ID，EX，MEM，WB 所用的控制信号，EX/MEM 需要存储 EX，MEM,WB 所用的控制信号，MEM/WB 需要存储 MEM，WB 需要的控制信号。

传递信号如图：



2.1.2 编写代码

定义各个存储控制信号的寄存器变量，由于各种变量名称极为复杂，所以这里采用 **“源寄存器_目标寄存器_信号名称”**的方式，源与目标针对的是信号的传递方向。

部分代码：

```
//for EX to MEM
reg [31:0] EX_MEM_BranchAddr;
reg EX_MEM_ZERO;
reg [31:0] EX_MEM_ALURes;
reg [31:0] EX_MEM_ReadData2;
reg [4:0] EX_MEM_INST_HL;

//to MEM
reg EX_MEM_Branch;
reg EX_MEM_MemWrite;
reg EX_MEM_MemRead;


//to WB
reg EX_MEM_RegWrite;
reg EX_MEM_MemToReg;
```

（这里定义了保存从 EX 传递到 MEM,WB 的信号的寄存器）

- **模块连接：**

同 lab05 一样，我们需要利用 wire 将各个模块的输入输出端口连接起来，比如带符号扩展模块的输出可能是 ALU 模块的操作数输入。

部分代码：

```
//EX
assign EX_BranchAddr = (ID_EX_SIGNEXT << 2) + PC_Add4;
assign EX_INST_HL = ID_EX_RegDst ? ID_EX_INST_LOW : ID_EX_INST_HIGH;
assign EX_ALUInput2 = ID_EX_ALUSrc ? ID_EX_SIGNEXT : ID_EX_ReadData2;
```

- **实例化：**

注意模块端口与寄存器变量一一对应。

部分代码：

```
dataMemory d0(
    clk,
    EX_MEM_MemWrite,
    EX_MEM_MemRead,
    EX_MEM_ALURes,
    EX_MEM_ReadData2,
    MEM_ReadData
);
```

- **段寄存器之间的信号传递：**

在 lab06 中，需要在各个段寄存器之间传递信号的值。因此在更新 PC 的同时，还要将部分寄存器的值传给下一阶段的寄存器。

部分代码：

```
EX_MEM_BranchAddr <= EX_BranchAddr;
EX_MEM_ZERO <= EX_ALUZero;
EX_MEM_ALURes <= EX_ALURes;
EX_MEM_ReadData2 <= ID_EX_ReadData2;
EX_MEM_INST_HL <= EX_INST_HL;
EX_MEM_RegWrite <= ID_EX_RegWrite;
EX_MEM_MemToReg <= ID_EX_MemToReg;
EX_MEM_Branch <= ID_EX_Branch;
EX_MEM_MemWrite <= ID_EX_MemWrite;
EX_MEM_MemRead <= ID_EX_MemRead;
```

2. 2 控制冒险与数据冒险

2.2.1 数据冒险

数据冒险：当一条指令需要用到前面某条指令的结果，从而不能重叠执行时，就发生了数据冒险。比如 lw 指令后为 add 指令，且 add 指令的输入来自于 lw 刚刚写入的寄存器，由于指令流水线，add 指令取操作数时 lw 还没有写入寄存器，导致数据冒险。这里采用在产生数据冒险的指令之间插入空指令进行停顿的做法。

2.2.2 控制冒险

控制冒险：当流水线遇到分支指令和其他能够改变 PC 值的指令时，就会发生控制冒险，因为对于跳转指令，下一个指令的 PC 值并非原 PC+4,但是在指令流水线中，PC 默认 PC=PC+4，所以在编写代码时需要对分支指令进行辨别和处理。这里采用的方法是检测是否有跳转指令，如果有那么暂停流水线，直到 branch 的结果和真正的跳转之后的指令地址出现再继续流水线。

处理代码：

```
else if(EX_MEM_Branch)
begin
    IF_ID_INST <= 0;
    PC <= NewPC;
    EX_MEM_Branch <= 0;
    MEM_WB_ReadData <= MEM_ReadData;
    MEM_WB_ALURes <= EX_MEM_ALURes;
    MEM_WB_INST_HL <= EX_MEM_INST_HL;
    MEM_WB_RegWrite <= EX_MEM_RegWrite;
    MEM_WB_MemToReg <= EX_MEM_MemToReg;
end

else if(ID_EX_Branch)
begin
    IF_ID_INST <= 0;
    PC <= PC;
    EX_MEM_INST_HL <= EX_INST_HL;
    EX_MEM_BranchAddr <= EX_BranchAddr;
    EX_MEM_ZERO <= EX_ALUZero;
    EX_MEM_RegWrite <= ID_EX_RegWrite;
    EX_MEM_MemWrite <= ID_EX_MemWrite;
    EX_MEM_MemRead <= ID_EX_MemRead;
    EX_MEM_ALURes <= EX_ALURes;
    EX_MEM_ReadData2 <= ID_EX_ReadData2;
    EX_MEM_MemToReg <= ID_EX_MemToReg;
    EX_MEM_Branch <= ID_EX_Branch;


    MEM_WB_ReadData <= MEM_ReadData;
    MEM_WB_INST_HL <= EX_MEM_INST_HL;
    MEM_WB_RegWrite <= EX_MEM_RegWrite;
    MEM_WB_MemToReg <= EX_MEM_MemToReg;
    MEM_WB_ALURes <= EX_MEM_ALURes;
    ID_EX_Branch <= 0;
end
```

3. 仿真测试

- mem_data.txt：

mem_data.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001000
```

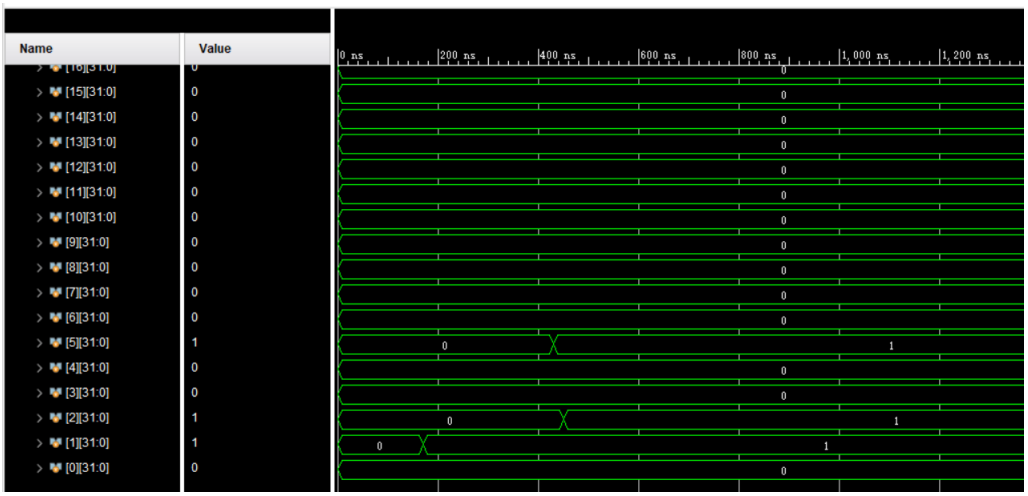
- mem_inst.txt

mem_inst.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
10001100000000010000000000000001//lw $1 1($0)
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000001000000010100000100000//add $5,$1,$0
10001100000000010000000000000001//lw $2,1($0)
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00010000001000100000000000000010//beq $1,$2,2
00000000001000100100100000100000//add $9,$1,$2
00000000000000000000000000000000
00000000000000000000000000000000
```

- **仿真结果：**



可以看出 5 号寄存器写入 1，避免了数据冒险，9 号寄存器无变化，说明 beq 跳转指令成功跳转，没有执行下一条 add 指令。仿真成功。

4. 感想

lab06 相较 lab05 难度又有了很大的提高，lab06 实现了指令的流水线，相当于性能得到了提高。在编写代码的过程中，我认识到严谨地做好每一步的重要性，不仅要思维严谨，变量定义和端口相连也要严谨，否则就会出错。在做实验的过程中，我对 cpu 的认识不再只局限于书本之上，当我真正实现了简单的 cpu 时，我才觉得自己真正地对计算机系统结构这部分的内容懂得了，掌握了。特别感谢老师和同学在实验中给予的教导和启发。

5. 参考文献

《cs145 实验指导书 lab06》