# 流水线设计思路

统一Clock

## **Step1：Fetch Instructions**

### 器件操作

Mux：选择指令

Add：PC + 4并写回PC为下一个时钟周期做好准备

指令寄存器：根据PC送来的地址进行取址

### Reg1：IF / ID – Store

1. 保存取址得到的指令
2. 32位PC自增地址（Add完的PC+4）

## **Step2：Decode**

### Reg1：IF / ID – Read

1. 将后16位立即数送入符号扩展单元
2. 将rs（旁路用）、rt、rd寄存器号送入寄存器堆

### 器件操作

每条指令在译码阶段都要判断

根据IF/ID寄存器传来的值进行读寄存器中的数据

对指令后16位进行符号扩展

冒险检测单元：插入阻塞

条件：P212

### Reg2：ID / EXE – Store

1. 寄存器读来的数据1
2. 寄存器读来的数据2
3. 符号扩展单元传来的扩展完的32位数据
4. PC自增值
5. 写寄存器传入ID/EX

## **Step3：Execution**

### Reg2：ID / EXE – Read

1. 寄存器1传来的数据送入ALU\_Source\_A
2. 寄存器2传来的数据送入ALU\_Source\_B\_Mux
3. 寄存器2传来的数据送入Reg\_EX/MEM
4. 符号扩展单元传来的数据送入左移单元
5. 符号扩展单元传来的数据送入ALU\_Source\_B\_Mux

### 器件操作

ALU：对Source\_A和Source\_B传来的数据进行相加

### Reg3：EXE / MEM

1. 将ALU\_Out\_Zero存入Reg\_EX/MEM
2. 将ALU\_Out\_C 存入 Reg\_EX/MEM
3. 写寄存器传入EX/MEM（from ID）

## **Step4：Memory Access**

### Reg3：EXE / MEM

1. 从Reg\_EX/MEM中得到的地址读取数据存储器（from ALU\_Out\_C）
2. 从Reg\_EX/MEM 中得到的数据写入地址对应的数据存储器（from RF\_Out\_2）

### 器件操作

数据存储器：根据传入的地址及数据进行写入 or 读取

### Reg4：MEM / WB

1. 将数据存储器中读出的数据写入Reg\_MEM/WB
2. 将ALU\_Out\_C的数据传入Reg\_MEM/WB
3. 写寄存器传入EX/MEM（from ID）

## **Step5：Write Back**

### Reg4：MEM / WB

1. 将从数据存储器中读出的数据传入RegWriter\_Mux
2. 将从ALU\_Out\_C的数据传入RegWriter\_Mux
3. 将写寄存器传入寄存器堆的写寄存器端口（from ID）

### 器件操作

寄存器堆：根据写寄存器地址及数据进行Write Back

## **Hazard**

### 结构冒险 Structural Hazard

#### Occasion

硬件资源竞争，IF阶段和MEM阶段同时进行内存访问

#### Solvation

指令、数据内存分开

### 数据冒险 Data Hazard

#### Occasion

上一个周期写回WB和这一个周期EX共用一个寄存器

#### Solvation

Forwarding，数据直接传递给下一个指令

1. 运算依赖，R-Type 🡪 R-Type：EX\_1\_Out to EX\_2\_In
2. 访存依赖，lw 🡪 R-Type：MEM\_1\_Out to EX\_2\_In，仍然需要nop来bubble一个周期

#### 旁路 & 阻塞

1. 旁路单元：判断冒险条件
   1. 当前面指令有写寄存器时才发生forwarding
   2. Rd≠$Zero
   3. 判断条件
      1. （A）EX/MEM.Register\_**Rd** = ID/EX.Register\_**Rs** or ID/EX.Register\_**Rt**

Forwarding from **EX/MEM.Register** to **ALUSource.Mux**

* + 1. （B）MEM/WB.Register\_**Rd** = ID/EX.Register\_**Rs** or ID/EX.Register\_**Rt**

Forwarding from **MEM/WB.Register** to **ALUSource.Mux**

1. 冒险检测单元：
   1. ID/EX.MemRead
   2. ID/EX.Register\_Rt = IF/ID.Register\_Rs
   3. ID/EX.Register\_Rt = IF/ID.Register\_Rt
   4. Output to：若满足则插入bubble，指令被延后阻塞一周期后，旁路单元可正常工作；处于ID级指令被阻塞，IF也被阻塞；保持PC、IF/ID.Register不变（插入nop）
      1. PCWrite（拒绝更新）
      2. IF/IDWrite
      3. ID/EX.Register.Mux（清空）

实现：

1. 强制ID/EX.Register = 0，则EX、MEM、WB为nop
2. 防止PC、IF/ID.Register更新（保持现状）🡪 一周期阻塞后可继续进行（lw要到EX）

### 控制冒险 Control Hazard

#### Occasion

决策依赖于一条指令的结果（beq/bne跳转地址取址地址在EXE后才知道）

#### Solvation\_1

Branch 🡪 Any：ID\_Special\_1\_Out to IF\_2（提到ID阶段计算跳转地址 & ID阶段减法直接判断，仍然需要nop来bubble一个周期

#### Solvation\_2

预测 / 有限状态自动机

## **流水线寄存器**

控制信号携带

IF/ID 🡪 Control单元解析 🡪 ID/EX （WB/MEM/EX）🡪 EX/MEM（WB/MEM）🡪 MEM/WB（WB）

旁路与阻塞

## **控制信号 P204 写入有效/无效时区=-0**

### Step1\_IF

读指令存储器和写PC的控制信号总是有效的

### Step2\_ID

### Step3\_EX

RegDst 选择结果寄存器

ALUOp ALU操作

ALUSrc 为ALU读取数据2或符号扩展后的立即数

### Step4\_MEM：

Branch相等则分支

MemRead 装载指令

MemWrite 存储指令

除非控制电路确定是一条分支指令并且ALU结果为0，否则将选择线性地址中的下一条指令作为PCSrc信号

### Step5\_WB

MemtoReg 决定将ALU结果or存储器数据传送到寄存器堆

RegWrite 决定是否写入寄存器堆