# U10M11007 试点班实验报告

## 三级流水线设计报告

姓 名: 王 嘉 利
学 号: 2018302278
班 号: 10011801

CS 11007 计算机组成与体系结构
(春季, 2020)

西北工业大学
计算机学院
ERCESI

2020 年 5 月 19 日

# 摘 要

本次实验在分析 SMIPA 指令集的基础上设计了三级流水处理器。将指令的执行分为取指、译码、执行访存写回三个阶段。设计了三级流水的数据通路和控制通路，分析了数据冒险、控制冒险和例外执行时的特征。最后列出了模块所需的接口。

# 版 权 声 明

# 目 录

# 1 概述

三级流水处理器将指令的执行分为三个阶段。在第一个阶段，PC 从指令内存中取出指令；第二阶段，将指令进行译码；第三阶段完成剩余的步骤。在出现冒险或者异常的情况下，流水线正常流，一拍一个指令。在处理跳转指令时，预测下一个指令就是 PC+4，并且将跳转目标的计算放在译码级，当出现控制冒险的时候，清空流水线从新的地址取值；由于三级流水的特点，简化了数据冒险；对于例外来讲，MIPS 是精确异常，异常指令前面的指令都处理结束，异常的提交是在流水线的最后一级，而且例外也可以看成一个控制冒险，流水线的清空都通过控制冒险单元完成。

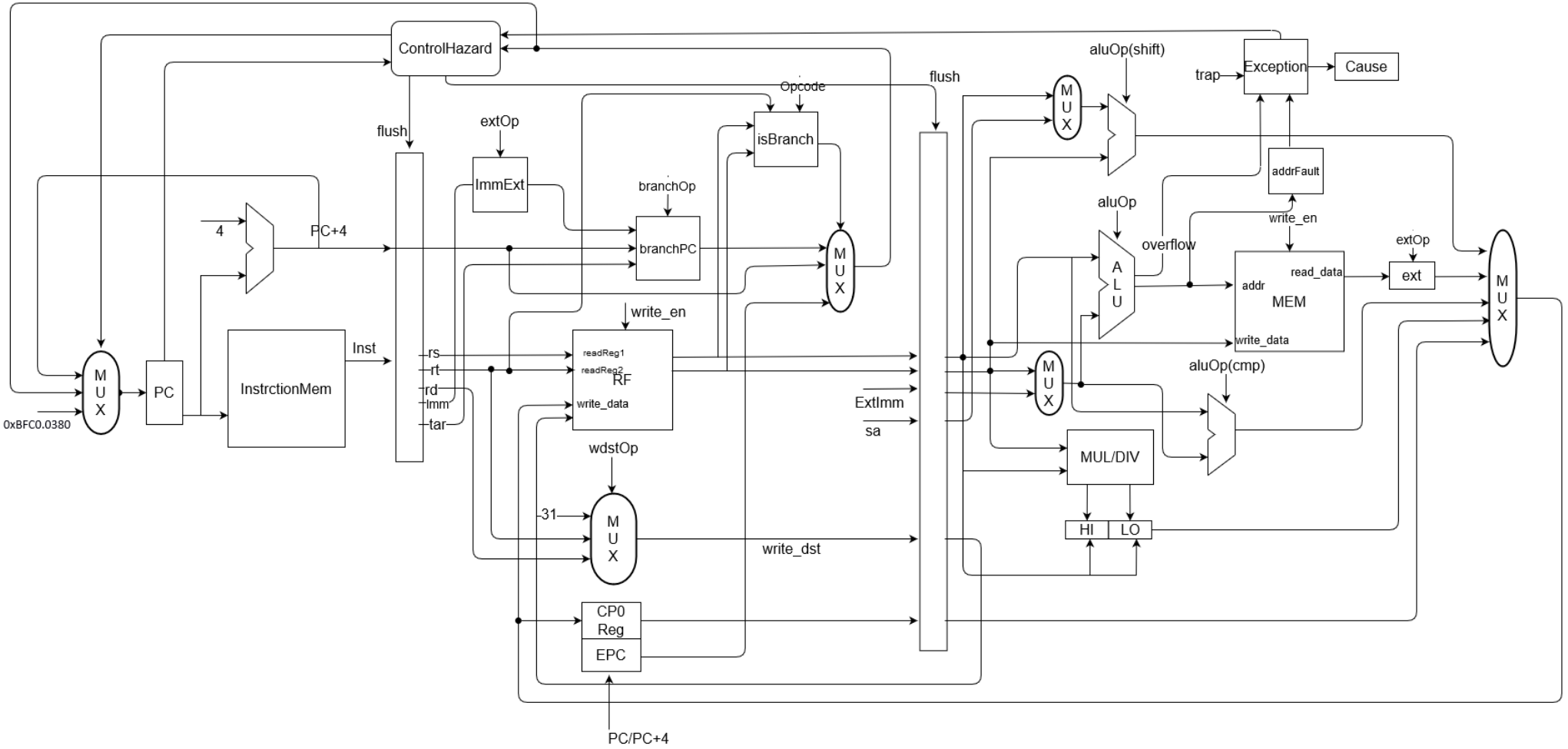


图 1: 三级流水处理器结构图

# 2 系统设计

## 2.1 System Overview

三级流水线处理器的结构图如上图所示

## 2.2 数据通路

- **取指** PC 从 InstructionMemory 从取出指令交给下一个阶段译码。而 PC 可能的取值是 PC+4、跳转的地址、异常处理的地址，也有可能来自 EPC，因此要用多选器选出 PC 的值，再从 InstructionMemory 中取出指令。但是在取值的时候并不知道 nextPC 是多少，因此，就采用静态分支预测，预测 nextPC 每次都是 PC+4
- **译码** 在这一阶段，需要将执行指令时可能用到的数据准备好，对立即数进行扩展等。另外，因为第三阶段的数据通路比较长，所以可以将判断是否跳转放到译码阶段，尽可能更早知道是否要跳转
- **执行访存和写回** 在执行指令时，可以把运算分为移位、算术逻辑运算、比较运算，以及乘除法；在访存时，访存的地址一定是来自算术运算的结果，写入的数据是 rt 的值；对于写回，写回寄存器的数据是来自执行和访存的输出，或者是 HiLo 寄存器。在译码时产生一信号，对写入的数据进行选择

## 2.3 控制通路

控制主要是通过`opcode func`解码出来的信号来进行控制的。每拍将相应的控制信号送到后面的流水线寄存器。对 ALU 来讲，控制比较有特点，因为将 ALU 分成了三个功能不同的 ALU，所以总体上可以直接将`opcode`段`func`段的后三位来进行控制关于 EPC 写入数据的选择信号来自 trap 信号（异常是否是自陷），如果是选择 PC+4，否则 PC。

## 2.4 冒险

### 2.4.1 数据冒险

考虑最简单的情况1, 第二条指令 h 和第一个指令相关，and 指令在译码阶段要用到 add 的执行结果，此时 RF 的写使能和和读是同时存在的，可以认为 RF 内部进行了 forward，消除了冒险。同理，可以发现，对一个寄存器的累加操作，即连续的`add $1,$1,$2`类似这样的操作，也不会出现冒险。再来考虑，需要的数据是 lowd 指令从 MEM 中取出来的，因为执行和访存是同一个阶段，因此也没有冒险。

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| `add $1,$2,$3` | F | D | EMW | | |
| `and $2,$1,$4` | | F | D | EMW | |
| `or $5,$1,$3` | | | F | D | EMW |

表 1: data hazard

可以发现在一般情况下，对于这种三级流水线都不会产生数据冒险

### 2.4.2 控制冒险和异常

控制冒险可以分成两类，一个是由于例外造成的，一个是由于分支预测错误造成的。

先来考虑由于分支预测错误造成的冒险，首先把是否冒险探测出来，因为将 nextPC 的计算提前，放到了译码阶段，因此通过检测取指的 PC 和译码时产生的 nextPC 是否相等就可以判断出是否有冒险产生。如果预测失败，那么就清空给取指的指令，PC 的值选择为 nextPC。这个通过一个单独的控制冒险单元实现。

再来考虑异常。首先考虑异常的产生情况，一种是自陷指令 (`break syscall`)，指令的目的就是为了产生异常，另一种是 ALU 算出来的结果溢出，或者是算出来的地址不正确。对于第一个，在译码阶段就可以发现，把这个异常信号记在流水线里，在最后一级流水线时提交。第二种异常，在第三级流水时产生，而且不希望错误的结果写回到寄存器或者存储器。用一个异常单元对异常进行检测和处理，处理异常时需要清空流水线，将异常原因写到 cause 寄存器，而且要根据不同的异常将 EPC 的值设为 PC 或者 PC+4。清空流

水线和和跳转到异常处理地址，可以通过控制冒险单元完成。

**e.g. 1**

```
add $1,$2,$3
beq $1,$2,100
lw $3, 0($4)
...
or $2,$1,$3 // 跳转目标
```

| | C1 | C2 | C3 |
|---|---|---|---|
| `add $1,$2,$3` | F | D | EMW |
| `beq $1,$2,100` | | F | D |
| `lw $3, 0($4)` | | | F |

(a) 探测到冒险

| | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|
| `add $1,$2,$3` | F | D | EMW | | |
| `beq $1,$2,100` | | F | D | EMW | |
| `lw $3, 0($4)` | | | F | buble | buble |
| `or $2,$1,$3` | | | | F | D |

(b) 处理冒险

**e.g. 2**

```
break
beq $1,$2, 100 // branch take
and $1,$2,$3
```

要考虑这种情况，当 break 进入最后一级流水时，发现了异常，ID 级流水同时也发现了分支预测错误。但是应该 ControlHazard 单元因该处理异常而不是分支预测错误。

```
if exception and nextPC != PC :
    PCSrc should choose ExceptionHandling address
```

# 3 模块详细设计

## 3.1 数据通路相关

### 3.1.1 ImmExt 扩展单元

#### 3.1.1.1 功能描述

对 16bits 立即数进行扩展

#### 3.1.1.2 接口定义

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| imm | 16 | in | 立即数 |
| extImm | 32 | out | 扩展后的立即数 |
| extOp | 1 | in | 符号还是无符号扩展 |

表 2: ImmExt 接口

#### 3.1.1.3 逻辑控制

可以发现,只有逻辑运算时才进行无符号扩展,进一步可以用`opcode[3]&opcode[2]`作为 extOp

| extOp | 0 | 1 |
|---|---|---|
| 扩展 | signed | unsigned |

### 3.1.2 branchPC

用来计算跳转的地址。branchOp 控制跳转地址的计算,通过`opcode[2:0]`产生这个信号。如果 `opcode[2:0]`是 0，2，3 输出是 target，否则是 extImm 与 PC 相加的结果。

### 3.1.3 isBranch

用来产生 next PC 的选择信号

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| extImm | 32 | in | 扩展后的立即数 |
| PC | 32 | in | pc |
| target | 32 | in | J，JR 等指令的跳转目标 |
| branchOp | 1 | in | 控制信号 |
| branchPC | 32 | out | 跳转目标 |

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| valA | 32 | in | R[rs] |
| valB | 32 | in | R[rt] |
| rt | 5 | in | branch 指令的 rt 段 |
| opcode | 6 | in | 控制信号 |
| PCSel | 2 | out | nextPC 的选择信号 |

### 3.1.4 Reg File

#### 3.1.4.1 功能描述

32 个通用寄存器堆，实现寄存器的读写

#### 3.1.4.2 接口定义

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| readReg1 | 5 | in | 读寄存器 1 |
| readReg2 | 5 | in | 读寄存器 2 |
| write_data | 32 | in | 写数据 |
| writeReg | 5 | in | 写寄存器 |
| write_en | 1 | in | 写使能 |
| read_data1 | 32 | out | 读数据 |
| read_data2 | 32 | out | 读数据 |

表 3: RF 接口

#### 3.1.4.3 逻辑控制

因为受到冒险和异常的影响，需要能够清空流水线，所以 write_en 的控制会较其他信号复杂一些当发现有控制冒险，或者在在执行指令时发现异常，应该让相应指令的 write_en 置零

### 3.1.5 ALU

#### 3.1.5.1 功能描述

- **算术、逻辑运算 ALU** 用来进行算术运算与逻辑运算,处理类似`ADD AND`这类指令
- **移位运算 ALU** 用来进行移位运算，处理类似于`SLL SRL` 这类指令
- **比较运算 ALU** 用来进行比较, 处理类似于`SLT`这类指令

#### 3.1.5.2 接口定义

**算术、逻辑运算 ALU**

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| alu_A | 32 | in | ALU 的操作数 |
| alu_B | 32 | in | ALU 的操作数 |
| aluOp | 3 | in | ALU 控制信号 |
| alu_out | 32 | out | ALU 计算输出 |
| overflow | 1 | out | 整数溢出标志 |

表 4: ALU 接口

**移位运算 ALU 比较运算 ALU** 剩下两个 ALU 模块的接口是一样的

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| alu_A | 32 | in | ALU 的操作数 |
| alu_B | 32 | in | ALU 的操作数 |
| aluOp | 3 | in | ALU 控制信号 |
| alu_out | 32 | out | ALU 计算输出 |

表 5: ALU 接口

#### 3.1.5.3 逻辑控制

**算术、逻辑运算 ALU**

| aluOp | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 运算 | ADD | ADDU | SUB | SUBU | AND | OR | NOR | XOR |

表 6: ALU 控制

ALU 的两个操作数一个是来自 rs,一个来自 rt 或者立即数。用`opcode[5:3]`来选择

```
if opcode[5:3] == 000 :
    alu_B = R[rt]
else :
    alu_B = extImm
```

**移位运算 ALU**

| aluOp | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 运算 | SLL | xxx | SRL | SRA | SLLV | xxx | SRLV | SRAV |

表 7: ALU 控制 (xxx 是不关心的情况)

这些移位指令都是 R-type 指令，ALU 的操作数 alu_B 来自 rt，alu_A 来自 R[rs] 或者 sa

```
alu_A = func[3] ? R[rt] : sa
```

**比较运算 ALU**

| aluOp | 010 | 011 | other |
|---|---|---|---|
| 运算 | signed cmp | unsigned cmp | xxx |

表 8: ALU 控制 (xxx 是不关心的情况)

ALU 的操作数和算术逻辑 ALU 的操作数一致

#### 3.1.6 InstMEM MEM

##### 3.1.6.1 指令内存

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| inst_sram_en | 1 | in | 使能信号 |
| inst_sram_wen | 4 | in | 写使能 |
| inst_sram_addr | 32 | in | 读写地址 |
| inst_sram_wdata | 32 | in | 写数据 |
| inst_sram_rdata | 32 | out | 写数据 |

##### 3.1.6.2 数据内存

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| data_sram_en | 1 | in | 使能信号 |
| data_sram_wen | 4 | in | 写使能 |
| data_sram_addr | 32 | in | 读写地址 |
| data_sram_wdata | 32 | in | 写数据 |
| data_sram_rdata | 32 | out | 写数据 |

### 3.2 控制通路相关

#### 3.2.1 ControlHazard

探测和处理控制冒险

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| PC | 32 | in | PC 寄存器中的值 |
| nextPC | 32 | in | 计算出的 nextPC |
| exception | 1 | in | 异常标志 |
| PCSrc | 2 | out | PC 的选择信号 |
| IF_flush | 1 | out | 清空 IF 信号 |
| ID_flush | 1 | out | 清空 ID 信号 |

清空 ID 流水可以将写信号置零，对于 IF 则插入一个 nop 指令

#### 3.2.2 Exception

对异常进行探测和处理

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| trap | 1 | in | 自陷指令异常标志 |
| overflow | 1 | in | 整数溢出异常标志 |
| addrFault | 1 | in | 地址错异常标志 |
| cause | 32 | out | 写到 Cause 寄存器的原因 |
| exception | 1 | out | 异常标志 |

addrFault 信号来自判读是否地址错单元的输出。

### 3.3 流水级顶层 module

#### 3.3.1 IF

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| nextPC | 32 | in | 计算出的 nextPC |
| PCSRc | 2 | in | PC 选择信号 |
| flush | 1 | in | 清空信号 |
| IM_en | 1 | in | instMem 使能 |
| PC | 32 | out | PC 值 |
| PC4 | 32 | out | PC+4 |
| Inst | 32 | out | 指令 |

表 9: IF 接口

#### 3.3.2 ID

因为考虑后面的流水级可能要使用关于 Inst 的信息，所以直接将 Inst 传到流水线寄存器。而且对于一些信号是要从第三级流水拉回来，同名的信号也会在译码产生，用 in 和 out 后缀来区分。
必要的控制信号也要输出

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| Inst | 32 | in | 指令 |
| PC4_in(out) | 32 | in(out) | PC+4 |
| PC_in(out) | 32 | in(out) | PC |
| write_dst_in(out) | 5 | in(out) | 寄存器目标 |
| write_reg_in(out) | 1 | in(out) | RF 写使能 |
| write_reg_data | 32 | in | 写会寄存器的值 |
| write_epc | 1 | in | 写 EPC 使能 |
| write_cp0reg_in(out) | 1 | in(out) | 写 CP0 寄存器使能 |
| flush | 1 | in | 清空信号 |
| reg_data1 | 32 | out | 寄存器读数据 |
| reg_data2 | 32 | out | 寄存器读数据 |
| extImm | 32 | out | 扩展后的立即数 |
| nextPC | 32 | out | 下一个 PC |
| Inst | 32 | out | 指令 |
| write_hilo | 1 | out | HI LO 写使能 |
| write_mem | 4 | out | 写 DMem 使能 |
| trap | 1 | out | 是否出现自陷指令 |
| extOp | 2 | out | MEM 读出数据后的扩展控制信号 |
| write_data_src | 3 | out | 写会数据的选择信号 |

表 10: ID 接口

### 3.3.3 EMW

EMW 的输入基本都来自于 ID 级译码的结果

| 名称 | 位宽 | 方向 | 描述 |
|---|---|---|---|
| Inst | 32 | in | 指令 |
| PC4_in(out) | 32 | in(out) | PC+4 |
| PC_in(out) | 32 | in(out) | PC |
| reg_data1 | 32 | in | 寄存器读数据 |
| reg_data2 | 32 | in | 寄存器读数据 |
| write_hilo | 1 | in | HI LO 写使能 |
| write_mem | 4 | in | 写 DMem 使能 |
| DM_en | 1 | in | DMem 使能 |
| trap | 1 | in | 是否出现自陷指令 |
| extOp | 2 | in | MEM 读出数据后的扩展控制信号 |
| write_data_src | 3 | in | 写回数据选择信号 |
| write_reg_data | 32 | out | 写回寄存器的值 |
| exception | 1 | out | 是否有例外 |

表 11: EMW 接口

# 参考文献

MK.Computer.Organization.and.Design.5th.Edition