# 计算机组成原理 P3 实验报告

彭杰奇 15061169

## 一、 模块定义

### 1. IFU 模块定义

（1）基本描述

IFU 主要功能是完成取指令功能。IFU 内部包括了 PC、IM(指令存储器)以及其他相关逻辑。IFU 除了能执行顺序取指令外，还能根据 PCSrc 反映的 beq 指令的执行情况来决定是顺序取指令还是转移取指令。

（2）模块接口

表 1 IFU 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| PCSrc | I | 当前指令是否是 beq 指令<br>1:当前指令是 beq 指令<br>0:当前指令不是 beq 指令 |
| Imm[31:0] | I | 若为 beq 指令，需要进行移位计算的立即数 |
| Reset | I | 复位信号，1:有效，0:无效 |
| Clk | I | 时钟信号 |
| Instr[31:0] | O | 当前指令输出 |

（3）功能定义

表 2 IFU 功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 复位 | 复位信号有效时，PC 设置为 0x00000000 |
| 2 | 取指令 | 根据 PC 当前值从 IM 中取指令输出 |
| 3 | 计算下一条指令地址 | beq == 0 时，PC ← PC + 4；<br>beq == 1 时，PC ← PC + 4 + Imm\|\|0^2 |

### 2. GRF 模块定义

（1）基本描述

GRF 模块为通用寄存器堆，主要由 32 个具有写使能端的 32 位寄存器组成，能够同时根据由 R1 和 R2 输入的地址从其中两个寄存器中读出数据，并根据 WR 中输入的地址向其中一个寄存器写入数据。

（2）模块接口

表 3 GRF 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| R1[4:0] | I | rs 寄存器的地址 |
| R2[4:0] | I | rt 寄存器的地址 |
| WR[4:0] | I | 要写入的寄存器的地址 |
| WriteData[31:0] | I | 要写入的数据 |
| Clk | I | 时钟信号 |
| RegWrite | I | 写使能信号，1:有效，0:无效 |
| RD1[31:0] | O | rs 寄存器的值 |
| RD2[31:0] | O | rt 寄存器的值 |

（3）功能定义

表 4 GRF 功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 读数据 | RD1←(R1)<br>RD2←(R2) |
| 2 | 写数据 | RegWrite 有效时，(WR)←WriteData |

## 3. ALU 模块定义

（1）基本描述

ALU 为算数逻辑单元，可以对输入的两个数据进行加、减、按位与和按位或操作，并能够判断输入数据是否相等。

（2）模块接口

表 5 ALU 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| A1[31:0] | I | 第一个运算数 |
| A2[31:0] | I | 第二个运算数 |
| ALUCtr[1:0] | I | ALU 控制信号<br>2’b00:加法运算<br>2’b01:减法运算<br>2’b10:按位与运算<br>2’b11:按位或运算 |
| Zero | O | 输入数据是否相等<br>1:相等<br>2:不相等 |
| ALUResult[31:0] | O | ALU 运算结果 |

（3）功能定义

表 6 ALU 功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 加法运算 | ALUResult ← A1+A2 |
| 2 | 减法运算 | ALUResult ← A1-A2 |
| 3 | 按位与运算 | ALUResult ← A1&A2 |
| 4 | 按位或运算 | ALUResult ← A1\|A2 |
| 5 | 等于判断 | Zero ← (A1-A2)==0?1:0 |

## 4. DM 模块定义

（1）基本描述

DM 模块为数据存储器，由一个 32bit * 32 字的 RAM 构成，起始地址为 0x00000000 用于存储数据。

（2）模块接口

表 7 DM 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Addr[6:2] | I | 读/写 DM 的地址 |
| WriteData[31:0] | I | 要写入 DM 的数据 |
| MemWrite | I | 写 DM 的控制信号 |
| MemRead | I | 读 DM 的控制信号 |
| ReadData[31:0] | O | 从 DM 读出的数据 |

（3）功能定义

表 8 DM 功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 读数据 | 当 MemRead 为 1 时，ReadData ← RAM(Addr) |
| 2 | 写数据 | 当 MemWrite 为 1 时，RAM(Addr) ← WriteData |

## 5. EXT 模块定义

（1）基本描述

EXT 模块的作用是将 16 位立即数扩展为 32 位。

（2）模块接口

表 9 EXT 模块接口

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| Imm_16[15:0] | I | 要扩展的 16 位立即数 |
| ExtOp[1:0] | I | 扩展方式选择信号<br>2’b00:符号扩展<br>2’b01:后接 16 位 0<br>2’b10:无符号扩展 |
| Imm_32[31:0] | O | 扩展后的 32 位立即数 |

（3）功能定义

表 10 EXT 功能定义

| 序号 | 功能名称 | 功能描述 |
| --- | --- | --- |
| 1 | 位数扩展 | ExtOp 为 2’b00 时，16 位立即数正常符号扩展为 32 位<br>ExtOp 为 2’b01 时，16 为立即数后接 16 位 0 扩展为 32 位<br>ExtOp 为 2’b10 时，16 为立即数无符号扩展为 32 位 |

# 二、 控制器设计

## 1. Controller 模块定义

（1）基本描述

Controller 模块为 CPU 控制器，可以根据输入指令的 opcode 和 funct 值输出各种控制信号。

（2）模块接口

表 11 Controller 模块接口

| 信号名 | 方向 | 描述 |
| --- | --- | --- |
| opcode[5:0] | I | 指令中的 opcode,即[31:26]位 |
| Funct[5:0] | I | 指令中的 funct,即[5:0]位 |
| RegDst | O | 寄存器写入端地址控制<br>1:选择 rd 字段<br>0:选择 rt 字段 |
| ALUSrc | O | ALU 输入端 A2 选择<br>1:选择 Imm_32<br>0:选择 RD2 |
| MemtoReg | O | 寄存器堆写入端 WD 选择<br>0:来自 ALU 输出<br>1:来自 DM 输出 |
| RegWrite | O | 写寄存器控制信号 |
| MemWrite | O | 写 DM 控制信号 |
| MemRead | O | 读 DM 控制信号 |
| ExtOp[1:0] | O | EXT 扩展方式控制信号 |

| | | |
|---|---|---|
| nPC_Sel | O | 判断是否为 beq 指令 |
| ALUCtr[1:0] | O | ALU 控制信号 |

（3）功能定义

表 12 Controller 功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | addu 指令 | 当前指令为 addu 时，RegDst、RegWrite 信号为 1，其他全为 0 |
| 2 | subu 指令 | 当前指令为 subu 时，RegDst、RegWrite、ALUCtr[1]信号为 1，其他全为 0 |
| 3 | ori 指令 | 当前指令为 ori 时，ALUSrc、RegWrite、ALUCtr[0]、ALUCtr[1]、ExtOp[1]信号为 1，其他全为 0 |
| 4 | lw 指令 | 当前指令为 lw 时，ALUSrc、MemtoReg、RegWrite、MemRead 信号为 1，其他全为 0 |
| 5 | sw 指令 | 当前指令为 sw 时，ALUSrc、MemWrite 信号为 1，其他全为 0 |
| 6 | beq 指令 | 当前指令为 beq 时，nPC_Sel、ALUCtr[1]信号为 1，其他全为 0 |
| 7 | lui 指令 | 当前指令为 lui 时，ALUSrc、RegWrite、ExtOp[0]信号为 1，其他全为 0 |

## 2. Controller 真值表

表 13 Controller 真值表

| Instr | | addu | subu | ori | lw | sw | beq | lui |
|---|---|---|---|---|---|---|---|---|
| opcode | I | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| funct | | 100001 | 100011 | N/A | | | | |
| RegDst | O | 1 | 1 | 0 | 0 | X | X | 0 |
| ALUSrc | | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| MemtoReg | | 0 | 0 | 0 | 1 | X | X | 0 |
| RegWrite | | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| nPC_Sel | | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ExtOp[1] | | X | X | 1 | 0 | 0 | 0 | 0 |
| ExtOp[0] | | X | X | 0 | 0 | 0 | 0 | 1 |
| MemRead | | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| MemWrite | | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ALUOp[1] | | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| ALUOp[0] | | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| ALUCtr[1] | O | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ALUCtr[0] | | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

## 3. ALU 控制器设计

ALU 控制器是控制器的一部分，它利用 funct[5:0]的值和过渡信号

ALUOp[1:0]得到控制 ALU 运算的 ALU 控制信号 ALUCtr[1:0]

表 14　ALU 控制器真值表

| ALUOp | Funct | ALUCtr | 运算 | 描述 |
|---|---|---|---|---|
| 00 | X | 00 | 加法 | 针对 lw、sw、lui 指令 |
| 01 | X | 01 | 减法 | 针对 beq 指令 |
| 11 | X | 11 | 按位或 | 针对 ori 指令 |
| 10 | 100001(addu) | 00 | 加法 | 针对 addu 指令 |
| 10 | 100011(subu) | 01 | 减法 | 针对 subu 指令 |

# 三、 测试程序设计

```
ori $1,$0,20
lui $2,1
addu $0,$1,$1
subu $3,$2,$1
addu $4,$3,$0
beq $3,$2,target
target:
   ori $5,$0,100
   ori $6,$3,200
   lui $6,0x0f0e
   addu $7,$6,$5
   sw $7,0($5)
   sw $6,4($5)
   subu $8,$7,$6
   lw $9,($5)
   lw $10,4($5)
```

预期结果

$at 0x00000014
$v0: 0x00010000
$v1: 0x0000ffec
$a0: 0x0000ffec
$a1: 0x00000064
$a2: 0x0f0e0000
$a3: 0x0f0e0064
$t0: 0x00000064
$t1: 0x0f0e0064
$t2: 0x0f0e0000
Mem[4]: 0x0f0e0064
Mem[8]: 0x0f0e0000

## 四、 思考题

1. 在上个学年的计组课程中，PC（程序计数器）位数被规定为 30 位，试分析其与 32 位 PC 的优劣。

PC（程序计数器）位数被规定为 30 位，和 32 位相比其实是省略了最后两位 0，这样 PC 在计算下一条指令的时候，只需要 PC+1 或者 PC+1+Imm，而且 beq 指令的立即数也无需左移 2 位，因而相对来说更加简单；而 32 位 PC 计算下一条指令需要 PC+4，与 MIPS 实际是相符的，方便直接应用，但是计数器是地址 4 倍，计算下一条指令没那么方便。

2. 现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

合理。

IM 使用 ROM，ROM 是只读存储器，使用它存储指令，可以避免在运行过程中误操作造成指令更改；

DM 使用 RAM，RAM 是随机存储器，可以写入也可以读出数据，满足 DM 的要求，而且有读写使能端，能够防止对数据的非法操作；

GRF 使用寄存器堆，可以满足存储和读出指定地址的数据，且寄存器具有使能端，能够满足 GRF 的写使能。

3. 结合上文给出的样例真值表，给出 RegDst， ALUSrc， MemtoReg，RegWrite, nPC_Sel, ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

RegDst=~op5&~op4&~op3&~op2&~op1&~op0

ALUSrc=~(~op5&~op4&~op3&~op2&~op1&~op0)&~(~op5&~op4&~op3&op2&~op1&~op0)

MemtoReg=op5&~op4&~op3&~op2&op1&op0

RegWrite=~(op5&~op4&op3&~op2&op1&op0)&~(~op5&~op4&~op3&op2&~op1&~op0)

MemWrite=op5&~op4&op3&~op2&op1&op0

nPC_sel=~op5&~op4&~op3&op2&~op1&~op0

ExtOp=~(~op5&~op4&op3&op2&~op1&op0)

4. 充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式，请给出化简后的形式。

RegDst=~op5&~op4&~op3&~op1&~op0 (110001)

ALUSrc=op5|op4|op3|op1|op0

MemtoReg=op5&~op4&~op2&op1&op0 (000110)

RegWrite=~(op5&~op4&op3&~op2&op1&op0)&~(~op5&~op4&~op3&op2&~op1&~op0)

MemWrite=op5&~op4&op3&~op2&op1&op0

nPC_sel=~op5&~op4&~op3&op2&~op1&~op0

ExtOp=op5|op4|~op3|~op2|op1|~op0

5. 事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

因为 nop 空指令机器码是 0x00000000，它除了使 PC+4 之外不起到其他作用，在 Controller 的控制信号真值表中，他对所有输出端口都是 X，X 可以看做 0 或 1 任意一个，所以没有必要加入真值表中。

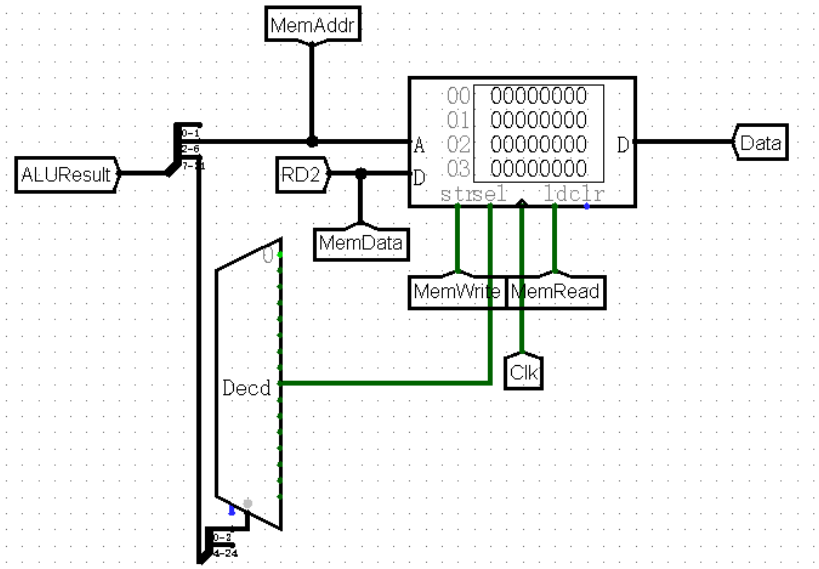
6. 前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM 片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

因为我们输入 IM 的指令是从 ROM 的 0 位地址开始的，这样在遇到 J 指令时可能出现问题，我们在 MARS 上写 MIPS 代码，如果把指令设在 0x00000300 开始，数据在 0x00000000 开始，DM 没有问题，beq 指令因为是相对寻址也没有问题，但是 J 指令是绝对寻址，就可能超出 IM 地址，所以我们要在 MARS 中设置指令在 0x00000000 开始，而数据段就要从 0x00002000 开始，我们原本用的 DM 是 32 位*32 字的，地址位从 0x00000000 开始，只到了 0x00000020,而我们不需要进行位扩展，因此只需要使用现在的 32*32 位的 DM 进行字扩展然后通

过片选信号选择我们现在使用的这片，模拟是从 0x0000200 开始的地址就好

0x00002000 转换成总共是 2^16 位，现在我们每个字单位是 32 位，所以总共是 2^8 个字单位，我们现在用的 DM 是 32 个也就是 2^5 个字单位，所以进行字扩展总共需要 2^8/2^5=8 个 DM,又因为我们的地址是要从 0x00002000 往后延伸，所以我们需要更多的 DM，所以就把地址延伸到 0x00004000，这样是 2^9 个字单位，需要 16 个 DM，然后我们要选取的就是第 9 个 DM，那就把我们现在的 DM 安排到第 9 个，当片选信号选择第 9 个时就可以了。这就相当于把 32*32 的存储器扩展为 512*32 的存储器并选其中第 9 个 DM

我们需要 9 位地址管脚，5 位连接 DM 地址，4 位产生 16 个片选信号，我们需要选择第 9 个，1000。




7. 除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。

形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了模拟验证的不足,验证时间短，有利于尽早尽快地发现和改正电路错误，缩短设计周期；但是形式验证不能有效验证电路的性能，如电路的时延和功耗。