# 单周期CPU实现报告

## 控制信号设计

```
typedef enum Vec2 {
    gprWriteRegisterSrc_rt,
    gprWriteRegisterSrc_rd,
    gprWriteRegisterSrc_ra
} GprWriteRegisterSrc;

typedef enum Vec2{
    gprWriteInputSrc_aluResult,
    gprWriteInputSrc_dmResult,
    gprWriteInputSrc_imm16
} GprWriteInputSrc;

typedef enum logic{
    aluInput1Src_gpr1,
    aluInput1Src_pc
} AluInput1Src;

typedef enum Vec2{
    aluInput2Src_gpr2,
    aluInput2Src_ext,
    aluInput2Src_4
}AluInput2Src;

typedef enum Vec2{
    pcJumpModeSrc_next=2'b00,
    pcJumpModeSrc_beq=2'b01,
    pcJumpModeSrc_abs=2'b10,
    pcJumpModeSrc_absreg=2'b11
}PcJumpModeSrc;

typedef enum Vec2{
    pcJumpInputSrc_0,
    pcJumpInputSrc_ext,
    pcJumpInputSrc_imm26,
    pcJumpInputSrc_gpr
}PcJumpInputSrc;

typedef struct packed{
    GprWriteRegisterSrc gprWriteRegisterSrc;
    GprWriteInputSrc gprWriteInputSrc;
    logic gprWriteEnabled;
    AluInput1Src aluInput1Src;
    AluInput2Src aluInput2Src;
    AluOp aluOp;
    logic dmWriteEnabled;
    logic extSign;
    PcJumpModeSrc pcJumpModeSrc;
    PcJumpInputSrc pcJumpInputSrc;
}ControlSignal;
```

| | gpr.readRegister1 | gpr.readRegister2 | gpr.writeRegister | gpr.writeInput | gpr.writeEnabled | alu.A | alu.B | alu.Op | dm.address | dm.writeInput | dm.writeEnabled | ext.input | ext.signed | pc.jumpMode | pc.jumpInput |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| addu | rs | rt | rd | alu.C | 1 | gpr.readData1 | gpr.readData2 | addu | | | | | | 00 | 0 |
| subu | rs | rt | rd | alu.C | 1 | gpr.readData1 | gpr.readData2 | subu | | | | | | 00 | 0 |
| ori | rs | | rt | alu.C | 1 | gpr.readData1 | ext.output | ori | | | | imm16 | 0 | 00 | 0 |
| lw | rs | | rt | dm.readResult | 1 | gpr.readData1 | ext.output | addu | alu.C | | 0 | imm16 | 1 | 00 | 0 |
| sw | rs | rt | | | 0 | gpr.readData1 | ext.output | addu | alu.C | gpr.readData2 | 1 | imm16 | 1 | 00 | 0 |
| beq | rs | rt | | | 0 | gpr.readData1 | gpr.readData2 | subu | | | | imm16 | 1 | {1'b0. alu.Zero} | ext.output |
| lui | | | rt | {imm16, 16{0}} | 1 | | | lui | | | | | | | |
| jal | | | 5'b11111 | alu.C | 1 | pc.pcValue | 8 | addu | | | | | | 01 | imm26 |
| jr | rs | | | | | | | jr | | | | | | 11 | gpr.readData1 |

# 控制信号描述

一共10个控制信号

## 无控制信号

例如通用寄存器组的输入，永远都只有一种可能输入，所以不需要放置控制信号，直接连线即可

## 通常控制信号

例如通用寄存器组的输出，有几种可能的输出就设计多大的控制信号，在后续用 case 赋值，如果是逻辑信号则直接连接。

## 特别的

- 引入了一个拓展器，拓展器接受一个控制信号区分 0 拓展和符号拓展，用来根据情况对 imm16 进行拓展
- 改写了 PC 模块，将本来的逻辑输入 jumpEnabled 改成了 2 位的 pcJumpMode，分别对应

```
                2'b00: pcReg <= pcReg + 4;
                2'b01: pcReg <= pcReg + 4 + {pcJumpInput[29:0], 2'b00};
// relative address
                2'b10: pcReg <= {pcReg[31:28], pcJumpInput[25:0], 2'b00}; //
absolute address
                2'b11: pcReg <= pcJumpInput;                                //
jump to address
```

  与之对应地接入控制信号，比较特别的是 beq 这种需要条件控制的，会在上层如下额外处理，值得注意的是这种写法可能不适合之后的 50 条指令

```
        if(controlSignal.pcJumpModeSrc == pcJumpModeSrc_beq)
            pcJumpMode = {1'b0, aluZero};
        else
            pcJumpMode = controlSignal.pcJumpModeSrc;
```

- aluOP 对于 R 型指令直接取 `funct`，否则手动指定