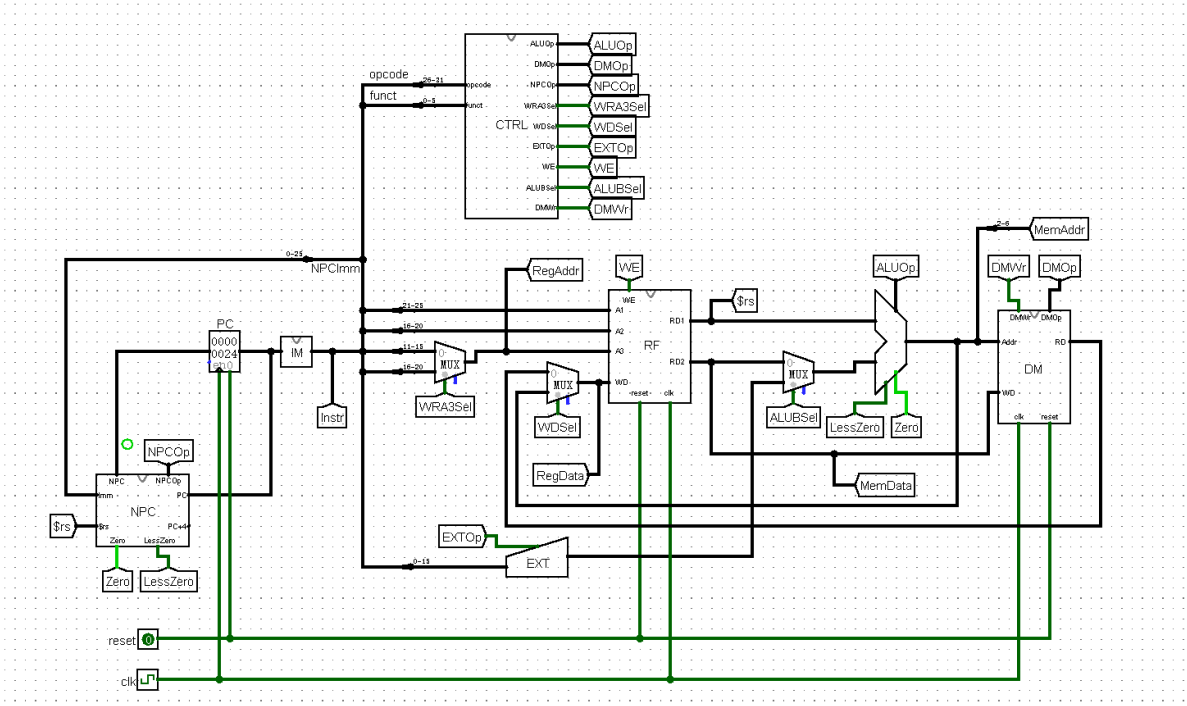
计算机组成原理实验报告—单周期CPU设计

总体设计概述

要求实现的指令集包括**addu，subu，ori，lw，sw，beq， jal，jr，lui，nop**

利用`Verilog`来实现时，参考了P3的Logisim电路设计，也做了一些改变

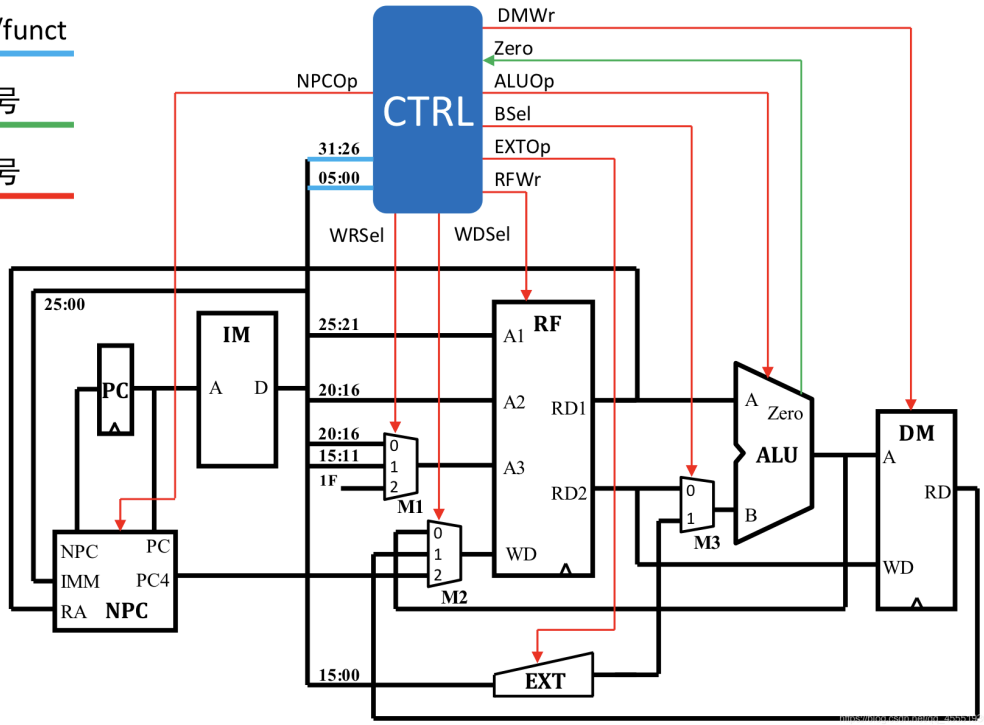


整体架构与高小鹏老师课件中的图类似

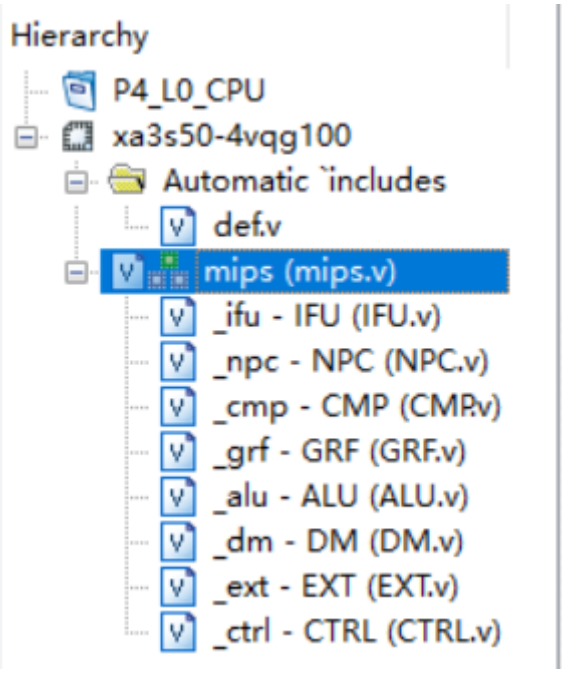
输入：opcode/funct

输入：状态信号

输出：控制信号



文件的树形结构层次如下



关键模块定义

变量名的命名遵从以下约定：

- 元件的命名为下划线开头的小写字母单词，例如`_grf`，`_alu`等
- 电线`wire`的命名为以下划线分隔的小写字母单词，例如ALU的32位输出命名为`alu_ans`，GRF的两个数据输出命名为`grf_rd1`和`grf_rd2`，判断是否是`addu`指令的电线命名为`addu`
- 控制信号如果关于元件的操作以`Op`结尾，如`ALUOp`，`DMOp`等；如果是多路选择器的控制信号，则以`Sel`结尾，如`GRFA3Sel`，`GRFWDSel`和`ALUBSel`等；如果是写使能信号则以`WrEn`结尾，如`GRFWrEn`，`DMWrEn`等
- 端口命名与P3相同
- 控制信号的宏定义命名为元件名+下划线+功能，如`ALU_add`，`NPC_jal_j`，`DM_w`，`DM_hu`等
- `opcode`和`funct`信号的宏命名为`op`/`fun`+下划线+大写指令名，如`op_ADDU`，`fun_ADDU`
- 所有的宏定义均包含在`def.v`文件内

GRF（寄存器堆）

端口说明

信号名称	方向	功能描述
A1[4:0]	输入	5位地址输入信号，将其储存的数据读出到RD1
A2[4:0]	输入	5位地址输入信号，将其储存的数据读出到RD2
A3[4:0]	输入	5位地址输入信号，将其作为写入数据的目标寄存器
RD1[31:0]	输出	输出A1指定的寄存器中的32位数据
RD2[31:0]	输出	输出A2指定的寄存器中的32位数据
WD[31:0]	输入	32位数据输入信号
GRFWrEn	输入	写使能信号；1：写入有效；0：写入无效
clk	输入	时钟信号
reset	输入	异步复位信号，将32个寄存器中的数据清零；1：复位；0：无效

控制信号说明

1. GRFA3Sel

控制信号值	功能
`A3Sel_rd`	选择待写入寄存器地址来自`Instr[15:11]`
`A3Sel_rt`	选择待写入寄存器地址来自`Instr[20:16]`
`A3Sel_ra`	选择写入寄存器的地址为31(`$ra`)

2. GRFWDSel

控制信号值	功能
`WDSel_dmrd`	选择写入寄存器的数据来自DM
`WDSel_aluans`	选择写入寄存器的数据来自ALU运算结果
`WDSel_pc4`	选择写入寄存器的数据为PC+4

IFU（取指单元）

把PC和IM合在一起形成了IFU

信号名称	方向	功能描述
NPC[31:0]	输入	待写入PC的指令地址
clk	输入	时钟信号
reset	输入	异步复位信号
PC	输出	当前指令地址
Instr[31:0]	输出	32位的指令值

EXT（位扩展）

将16位二进制数进行零扩展或符号扩展到32位

控制信号说明

控制信号值	功能
`EXT_unsign`	零扩展
`EXT_sign`	符号扩展

ALU（算术逻辑单元）

端口说明

信号名称	方向	功能描述
A[31:0]	输入	32位输入运算数A
B[31:0]	输入	32位输入运算数B
ALUOp[4:0]	输入	控制信号
shamt[4:0]	输入	移位指令所移位数
C[31:0]	输出	32位输出运算结果
overflow	输出	指示运算是否溢出（扩展用）

控制信号说明

1. ALUOp

控制信号值	功能
`ALU_add`	执行加法运算
`ALU_sub`	执行减法运算
`ALU_or`	执行逻辑或运算
`ALU_lui`	执行`lui`指令

2. ALUBSel

控制信号值	功能
`BSel_rt`	选择寄存器中的值进行运算
`BSel_imm`	选择立即数进行运算

CMP（比较器）

把原来ALU中比较值是否相等的运算移到了CMP里面，去指导`beq`这一类型的指令是否跳转

控制信号目前只有`CMP_beq`，未来可以扩展

```
1  `timescale 1ns / 1ps
2
3  `include "def.v"
4
5  module CMP(
6      input [31:0] rs,
7      input [31:0] rt,
8      input [2:0] CMPOp,
9      output jump
10     );
11
12     wire eq = (rs == rt);
13     wire ne = !eq;
14     wire le0 = (rs < 0);
15     wire ge0 = (rs > 0);
```

```
    wire eq0 = (rs == 0);

    assign jump = (CMPOp == `CMP_beq && eq) ? 1 : 0;

endmodule

```

端口说明

信号名称	方向	功能描述
rs[31:0]	输入	`$rs`寄存器的值
rt[31:0]	输入	`$rt`寄存器的值
CMPOp[2:0]	输入	控制信号
jump	输出	指示是否跳转

NPC（次地址计算单元）

把`beq`是否执行的判断交给了`CMP`，直接根据输入信号`jump`判断是否跳转

端口说明

信号名称	方向	功能描述
PC[31:0]	输入	32位输入当前地址
jump	输入	指示b类型指令是否跳转
NPCOp[1:0]	输入	控制信号
RA[31:0]	输入	`$ra`寄存器保存的32位地址
NPC[31:0]	输出	32位输出次地址
PC+4[31:0]	输出	输出PC+4的值

控制信号说明

控制信号值	功能
`NPC_pc4`	`NPC=PC+4`
`NPC_b`	执行`beq`指令
`NPC_j_jal`	执行`j`，`jal`指令
`NPC_jalr_jr`	执行`jalr`，`jr`指令

DM（数据储存器）

端口说明

信号名称	方向	功能描述
Addr[31:0]	输入	待操作的内存地址
WD[31:0]	输入	待写入内存的值
clk	输入	时钟信号
reset	输入	异步复位信号
DMWrEn	输入	写使能信号；1：写入有效；0：写入无效
DMOp[2:0]	输入	控制信号
RD[31:0]	输出	输入地址指向的内存中储存的值

控制信号说明

控制信号值	功能
`DM_w`	对应 `lw` 和 `sw` 指令，写入或读取整个字
`DM_h`	（保留）对应 `lh` 和 `sh` 指令，写入或读取半字
`DM_b`	（保留）对应 `lb` 和 `sb` 指令，写入或读取整个字
`DM_hu`	（保留）对应 `lhu` 指令
`DM_bu`	（保留）对应 `lbu` 指令

数据通路分析

CTRL同时承担了译码的任务

指令	opcode	funct	NPCOp	GRFA3Sel	GRFWDSel	EXTOp	GRFWrEn	ALUBSel	ALUOp	DMWrEn	DMOp
addu	000000	100001	`NPC_pc4`	`A3Sel_rd`	`WDSel_aluans`	X	1	`BSel_rt`	`ALU_add`	0	X
subu	000000	100011	`NPC_pc4`	`A3Sel_rd`	`WDSel_aluans`	X	1	`BSel_rt`	`ALU_sub`	0	X
ori	001101	X	`NPC_pc4`	`A3Sel_rt`	`WDSel_aluans`	`EXT_unsign`	1	`BSel_imm`	`ALU_or`	0	X
lw	100011	X	`NPC_pc4`	`A3Sel_rt`	`WDSel_dmrd`	`EXT_sign`	1	`BSel_imm`	`ALU_add`	0	`DM_w`
sw	101011	X	`NPC_pc4`	X	`WDSel_dmrd`	`EXT_sign`	0	`BSel_imm`	`ALU_add`	1	`DM_w`
beq	000100	X	001	X	X	X	0	X	X	0	X
lui	001111	X	`NPC_pc4`	`A3Sel_rt`	`WDSel_aluans`	X	1	`BSel_imm`	`ALU_lui`	0	X

调试过程记录

- 需要注意本次实验IM和DM均扩展到了4KB，因此输入地址位需要取 `[11:2]` 位

调试方法与辅助工具

采用随机程序生成和自动化测试的方法，支持的指令有**addu，subu，ori，lw，sw，beq， jal，lui，nop**

采用C语言生成随机程序，用 `iverilog` 和 `gtkwave` 编译模拟， `cmd` / `powershell` 命令行脚本自动化循环测试的方式

思考题

1. 根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

MIPS中以字节为单位，而在我们设计的DM中，每一个`reg[31:0]`为一个单位。

Addr来自ALU的输出端口，代表要读取的DM存储器的地址，在我们的4KB的DM设计中应当取[11:0]，又因为按字节寻址，因此取[11:2]

2. 思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣

三目运算符

```
assign ALUOp = (addu | lw | sw) ? `ALU_add :
               (subu) ? `ALU_sub :
               (ori) ? `ALU_or :
               (lui) ? `ALU_lui :
               4'b0000;
```

`case`语句

```
case(ALUOp)
    `ALU_add: C = A + B;
    `ALU_sub: C = A - B;
    `ALU_or: C = A | B;
    `ALU_and: C = A & B;
    `ALU_xor: C = A ^ B;
    `ALU_sll: C = B << shamt;
    `ALU_srl: C = B >> shamt;
    `ALU_sra: C = $signed($signed(B) >> shamt);
    `ALU_lui: C = B << 16;

    // Ready for add other instructions...

    default: C = 32'h0000_0000;
endcase
```

此外还可以用`if...else...`语句

- `assign`三目运算符不需要自己再定义寄存器
- `case`语句和`assign`都可以通过宏定义的方式使代码更加美观，增强可读性
- `if...else...`语句没用过

3. 在相应的部件中，**reset的优先级**比其他控制信号（不包括clk信号）都要**高**，且相应的设计都是**同步复位**。清零信号reset所驱动的部件具有什么共同特点？

都是存储器，例如`PC`、`GRF`和`DM`

4. C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。

根据RTL语言描述：`addi`与`addiu`的区别在于当出现溢出时，`addiu`忽略溢出，并将溢出的最高位舍弃；`addi`会报告`SignalException(IntegerOverflow)`

故忽略溢出，二者等价。

5. 根据自己的设计说明单周期处理器的优缺点。

- 优点：设计简单，扩展性好
- 缺点：时钟频率取决于执行时间最长的指令，整体时钟周期长，效率较低