





## 主要模块

主要模块包含COMMAND、GRF、ALU、IFU、DM、EXT、CONTROL

### COMMAND

- 在P4中添加，简化顶层设计

#### 端口说明

端口名称	方向	大小	说明
command	input	[31:0]	IFU模块输出的这一时钟周期内CPU执行的指令
rs	output	[4:0]	command[25:21]
rt	output	[4:0]	command[20:16]
rd	output	[4:0]	command[15:11]
funct	output	[5:0]	command[5:0]
opcode	output	[5:0]	command[31:26]
imm15	output	[15:0]	command[15:0]
imm25	output	[25:0]	command[25:0]

以下模块在logisim设计文档中已有对于端口的详细定义

在此仅作控制信号说明

#### GRF

#### 控制信号

### 1. GRF\_addr

选择对寄存器A3进行输入

GRF_addr	A3
0	rt
1	rd
2	31

### 2. GRF\_data

选择输入寄存器的值WD

GRF_data	WD
0	ALU_result
1	DM_out
2	PC+4

### 3. GRF\_WE

选择是否写入GRF

GRF_WE	Write?
0	√
1	×

## ALU

### 控制信号

#### 1. ALU\_src

选择ALU的运算数B

ALU_src	B
0	ALU_RD2
1	EXT_out

## 2. ALU\_op

选择ALU的运算R

ALU_op	R
0	add
1	sub
2	or

## IFU

### 控制信号

#### 1. PC\_op

选择下一条指令的PC计算方式（根据执行的指令C）

PC_op	C
1	beq
2	jal
3	jr

## DM

### 控制信号

#### 1. DM\_WE

选择是否写入DM

DM_WE	Write?
0	√
1	×

EXT

控制信号

1. EXT\_op

选择EXT的方式

EXT_op	WD
0	zero_ext
1	sign_ext
2	<<16

CONTROL

控制信号列表

	add	sub	ori	lw	sw	beq	lui	jal	jr
EXT_op	x	x	0	1	1	x	2	x	x
ALU_op	0	1	2	0	0	1	0	x	x
PC_op	0	0	0	0	0	1	0	2	3
DM_WE	0	0	0	0	1	0	0	0	0
GRF_WE	1	1	1	1	0	0	1	1	0
GRF_addr	1	1	0	0	x	x	0	2	x
GRF_data	0	0	0	1	x	x	0	2	x
ALU_src	0	0	1	1	1	0	1	x	x

思考题

---

1. 阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即  $32\text{bit} \times 1024\text{字}$ ），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0]？

文件	模块接口定义
dm.v	<pre>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</pre>

- DM模块中的寄存器寻址方式对应按字寻址，字地址addr即为所取的寄存器编号
  - addr信号由ALU模块的结果给出
  - 由于DM容量为4KB，地址应该输入10位信号；又因为LW、SW指令均按字节寻址，所以取10位的[11:2]而非[9:0]
2. 思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。
- 以指令sub、ori与信号ALU\_op、GRF\_WE为例

### 指令对应控制信号取值

```
if (sub) begin  
    ALU_op = 2'b01;  
    GRF_WE = 1'b1;  
    //...  
end  
if (ori) begin  
    ALU_op = 2'b10;  
    GRF_WF = 1'b1;  
    //...  
end
```

**\*\*控制信号取值对应指令**

```
ALU_op[0] = sub | ...;  
ALU_op[1] = ori | ...;  
GRF_op = sub | ori | ...;
```

- 第一种方式

优点：增添指令时不易错漏

缺点：增加信号时容易在某些指令中忽略；书写麻烦

- 第二种方式

优点：清晰易书写；美观；添加信号时不宜错漏

缺点：增加指令时容易忽略其部分信号

- 本人使用的第二种方式

3. 在相应的部件中，复位信号的设计都是**同步复位**，这与 P3 中的设计要求不同。请对比**同步复位**与**异步复位**这两种方式的 reset 信号与 clk 信号优先级的关系。

- 同步复位clk优先级更高

- 异步复位reset优先级更高

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。

- 以ADD与ADDU为例

ADD的Operation如下

```
temp ← (GPR[rs]31||GPR[rs]31..0) + (GPR[rt]31||GPR[rt]31..0) if temp32 ≠ temp31
then      SignalException(IntegerOverflow) else      GPR[rd] ← temp endif
```

ADDU的Operation如下

```
temp ← GPR[rs] + GPR[rt] GPR[rd] ← temp
```

- ADD在ADDU的操作之上加入了双符号位判断溢出的操作

- 若不考虑溢出，忽略ADD的溢出判断，则ADD与ADDU等价

## 测试样例

---