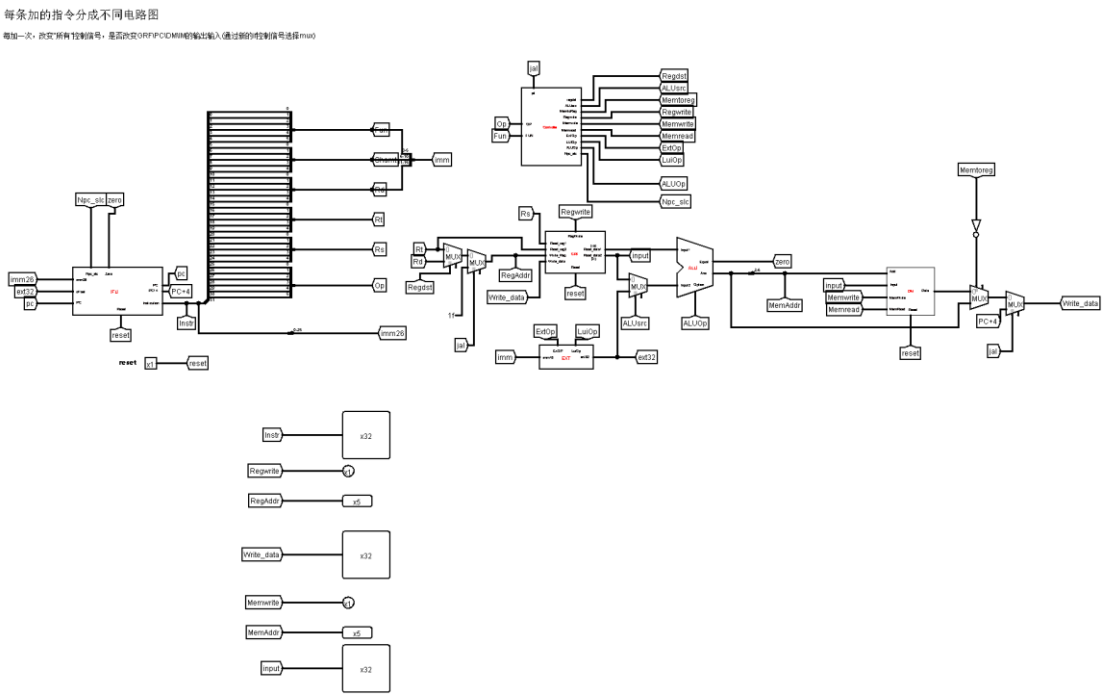
单周期 CPU 设计报告

一、数据通路设计

1、



二、模块规格

1. IFU

（1）端口说明

端口名	方向	描述
CLK	I	时钟信号
RESET	I	异步复位信号
PC_in	I	原 PC 信号
PC_out	O	PC+4 后的信号
INSTRUCTION[31:0]	O	输出 32 位指令

（2）功能定义

序号	功能名称	功能描述
1	复位	当 RESET 信号为 1 时，对保存 PC 地址的寄存器进行异步清零操作。
2	取指	以 PC 的值为地址，从 IM 中取出对应指令

2. GRF

（1）端口说明

端口名	方向	描述
CLK	I	时钟信号
RESET	I	异步复位信号
RegWrite	I	写入使能
Read_Reg1	I	读取 Rs
Read_Reg2	I	读取 Rt
Write_Reg	I	读取存储地址
Write_Data	I	读取存储数据
Read_Data	O	Rs 输出
Read_Data2	O	Rt 输出

（2）功能定义

序号	功能名称	功能描述
1	复位	当 RESET 信号为 1 时，对 GRF 中所有寄存器进行异步清零操作。
2	写入	Regwrite=1 时，将数据写入
3	读取	将 Rs，Rt 输出

3、ALU

（1）端口说明

端口名	方向	描述
Input1[31:0]	I	输入 1
Input2[31:0]	I	输入 2
ALUOp[31:0]	I	选择计算功能
Equ	O	输入是否相等
Ans[31:0]	O	输出

（2）功能定义

序号	功能名称	功能描述
1	逻辑运算	ALUOp=00，C=A&B ALUOp=01，C=A|B ALUOp=10，C=A+B ALUOp=11，C=A-B
2	相等判断	若 A=B，则 Equ=1 若 A≠B，则 Equ=0

4、EXT

（1）端口说明

端口名	方向	描述
Imm16[15:0]	I	16 位立即数输入
ExtOp	I	Ext 功能选择
LuiOp	I	Lui 功能选择
Ext32	O	32 位数输出

（2）功能定义

序号	功能名称	功能描述
1	符号拓展	ExOp=0，无符号拓展 ExOp=1，有符号拓展

2	Lui 指令功能	LuiOp=1，将立即数添加至 32 位数高位，低位补零

5、DM

（1）端口说明

端口名	方向	描述
CLK	I	时钟信号
RESET	I	异步复位信号
Addr[5:0]	I	地址读入
Input[31:0]	I	数据读入
MemWrite	I	写入使能
MemRead	I	读取使能
Data[31:0]	O	32 位数据输出

（2）功能定义

序号	功能名称	功能描述
1	写入	当写入使能=1 时，将 Input 写入 Addr 对应的寄存器
2	读取	当读取使能=1 时，将 Addr 对应的寄存器的值输出

6、Controller

（1）端口说明

端口名	方向	描述
OP	I	Operation
FUN	I	Function
Regdst	O	GRF 写入地址 MUX 控制 0：写入至 Rt 1：写入至 Rd
ALUsrc	O	ALU 输入 MUX 控制 0：input2 = Rt

		1：input2 = ext32
MemtoReg	O	GRF 写入数据 MUX 控制 0：写入 DM 的输出 1：直接输出 ALU 结果
RegWrite	O	GRF 写入使能
MemWrite	O	DM 写入使能
MemRead	O	DM 输出使能
ExtOp	O	Ext 使能信号
LuiOp	O	Lui 使能信号
BeqOp	O	Beq 使能信号
ALUOp[1:0]	O	ALU 功能选择信号

（2）功能定义

序号	功能名称	功能描述
1	控制	根据 Op 和 Fun 输出相应指令与 MUX 的控制信号

三、控制器设计

1. 端口说明

端口名	方向	描述
OP	I	Operation
FUN	I	Function
Regdst	O	GRF 写入地址 MUX 控制 0：写入至 Rt 1：写入至 Rd
ALUsrc	O	ALU 输入 MUX 控制

		0：input2 = Rt 1：input2 = ext32
MemtoReg	O	GRF 写入数据 MUX 控制 0：写入 DM 的输出 1：直接输出 ALU 结果
RegWrite	O	GRF 写入使能
MemWrite	O	DM 写入使能
MemRead	O	DM 输出使能
ExtOp	O	Ext 使能信号
LuiOp	O	Lui 使能信号
BeqOp	O	Beq 使能信号
ALUOp[1:0]	O	ALU 功能选择信号

2. 指令真值表

Instr	addu	subu	ori	lw	sw	beq	lui
Op	000000	000000	001101	100011	101011	000100	001111
Func	100001	100011	NA	NA	NA	NA	NA
Regdst	1	1	0	0	X	X	0
ALUsrc	0	0	1	1	1	0	1
MemtoReg	1	1	1	0	X	X	1
RegWrite	1	1	1	1	0	0	1

MemWrite	0	0	0	0	1	0	0
MemRead	0	0	0	1	0	0	0
ExtOp	X	X	0	1	1	0	X
LuiOp	X	X	0	0	0	0	1
BeqOp	X	X	X	X	X	1	X
ALUOp	10	11	01	10	10	XX	10

四、测试数据

A：测试 beq,ori,subu

```
ori $t0,1
ori $t1,1
beq $t0,$t1,jump
subu $t0,$t0,$t1
jump:
subu $s0,$s1,$t1
nop
```

期望输出：

```
@00003000: $ 8 <= 00000001
@00003004: $ 9 <= 00000001
@00003010: $16 <= ffffffff
```

B:测试 sw, lw, ori, addu

```
ori $t0,2
ori $t1,3
ori $t2,4
addu $s0,$zero,$zero
sw $t0,0($s0)
sw $t1,4($s0)
addu $s0,$t2,$s0
lw $t1,0($s0)
lw $t0,-4($s0)
```

期望输出：

```
@00003000: $ 8 <= 00000002
@00003004: $ 9 <= 00000003
@00003008: $10 <= 00000004
@0000300c: $16 <= 00000000
@00003010: *00000000 <= 00000002
@00003014: *00000004 <= 00000003
@00003018: $16 <= 00000004
@0000301c: $ 9 <= 00000003
@00003020: $ 8 <= 00000002
```

C：测试极端数据

```
lui $t0,65535
ori $t0,$t0,65535
```

期望输出：

```
@00003000: $ 8 <= ffff0000
@00003004: $ 8 <= ffffffff
```

D：测试 beq，j

```
ori $t1,$t1,1
beq_forward:
addu $t1,$t1,1
ori $t2,$t2,2
beq $t1,$t2,beq_forward    #3
ori $t3,$t3,2
j j_behind
addu $t3,$t3,1
j_behind:
subu $t3,$t3,2 #0
```

期望输出：

```
@00003000: $ 9 <= 00000001
@00003004: $ 1 <= 00000000
@00003008: $ 1 <= 00000001
```

```
@0000300c: $ 9 <= 00000002
@00003010: $10 <= 00000002
@00003004: $ 1 <= 00000000
@00003008: $ 1 <= 00000001
@0000300c: $ 9 <= 00000003
@00003010: $10 <= 00000002
@00003018: $11 <= 00000002
@0000302c: $ 1 <= 00000000
@00003030: $ 1 <= 00000002
@00003034: $11 <= 00000000
```

E：测试 jal，jr

```
ori $t0, $t0, 0
ori $t1, $t1, 1
ori $t3, $t3, 2
ori $t2, 8
ori $sp, $sp, 0x0100
lui $s0, 0
ori $a0, 10
jal fiber
jal end
fiber:
```

```
    beq  $a0, $t0, out0
    beq  $a0, $t1, out1
    subu $sp, $sp, $t2
    sw   $a0, 0($sp)
    sw   $ra, 4($sp)
    subu $a0, $a0, $t1
    jal  fiber
    lw   $a0, 0($sp)
    lw   $ra, 4($sp)
    addu $sp, $sp, $t2
    subu $sp, $sp, $t2
    sw   $a0, 0($sp)
    sw   $ra, 4($sp)
    subu $a0, $a0, $t3
    jal  fiber
    lw   $a0, 0($sp)
    lw   $ra, 4($sp)
    addu $sp, $sp, $t2
    jr   $ra
out1:
    addu $s0, $s0, $t1
    jr   $ra
```

```
out0:
    jr   $ra
end:
    addu $8, $s0, $0
```

期望输出：

```
    @00003000: $ 8 <= 00000000
    @00003004: $ 9 <= 00000001
    @00003008: $11 <= 00000002
    @0000300c: $10 <= 00000008
    @00003010: $29 <= 00000100
    @00003014: $16 <= 00000000
    @00003018: $ 4 <= 0000000a
    @0000301c: $31 <= 00003020
    @0000302c: $29 <= 000000f8
    @00003030: *000000f8 <= 0000000a
    @00003034: *000000fc <= 00003020
    @00003038: $ 4 <= 00000009
    @0000303c: $31 <= 00003040
    @0000302c: $29 <= 000000f0
    @00003030: *000000f0 <= 00000009
    @00003034: *000000f4 <= 00003040
    @00003038: $ 4 <= 00000008
```

@0000303c: $31 <= 00003040

@0000302c: $29 <= 000000e8

五、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？

答：

由于 MIPS 为 32 位，所以地址总为 4 的倍数，对 addr 需进行左移两位的操作，addr 来自 alu_out。

2、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 是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

答：

需对 IM,GPR,DM 清零。

IM 清零可清空已经加载的 instruction，为再一次加载做准备。

GPR 清零可清空寄存器的值，返回初始状态。

DM 清零可清空内存，返回初始状态。

3、列举出用 Verilog 语言设计控制器的几种编码方式（至少三种），并给出代码示例。

答：

（1）if_else 型：

```
always @(*) begin
    if()
    else if()
    else if()
    else()
end
```

（2）case 型：

```
always @(*) begin
    case()
        6’b000000:
        6’b000001:
        default:
    endcase
end
```

（3）assign 型：

```
assign lh    = (OpCode == 6'b100001);
assign lhu = (OpCode == 6'b100101);
assign lb    = (OpCode == 6'b100000);
assign RegDst = addu | subu | sll | slt | sra;
assign RegWrite = addu | subu | ori | lui | lw | jal | sll | slt | lh | lb | lhu | lbu |
sra | sltiu;
assign ALUSrc = ori | lui | lw | sw | lh | lhu | lb | lbu | sh | sb | sltiu;
assign MemWrite = sw | sb | sh;
```

（4）门级：

```
wire lh;
and (lh, opcode[5] , ~opcode[4] , ~opcode[3] , ~opcode[2] , ~opcode[1] ,
opcode[0]);
or ( RegDst , addu , subu , sll , slt , sra);
```

4、根据你所列举的编码方式，说明他们的优缺点。、

答：

assign 型和门级易于观测出每个信号的与或逻辑组成，而 if_else 型和 case 型则易于观测出每个指令直接对应的所有的信号变换，双方的优点即为对方的不足之处。

5、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 。

答：

由《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分可知，addi 的具体操作是对 rs 和 imm 进行符号扩展为 33 位再相加，如果不溢出，则将得到结果的低 32 位存入 rt 中，而 addiu 则是 imm 符号扩展至 32 位，并与 rs 进行相加存入 rt 中。因此，如果不考虑溢出，二者均是将结果的低 32 位存入 rt，等价。

同理，add 与 addu 也等价。

6、根据自己的设计说明单周期处理器的优缺点。

答：

优点在于：设计简单，易于实现，每个模块的建立于存在相对分离，易于搭建模块与模块之间的关系。

缺点在于：与流水线相比，单周期的 CPU 吞吐量较低，性能较低且成本较高。

7、简要说明 jal、jr 和堆栈的关系。

答：

当使用 jal 时，jal 跳转到被调用函数，并将此时 PC+4 的地址储存到堆栈中，即进行压栈保护，若在被调用的函数中使用 jr，则让堆栈中的地址出栈，jr 指令让 PC 值跳转回该地址所代表的位置