P4-Verilog 单周期实验报告

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 实现的单周期 MIPS - CPU，处理器为 32 位处理器，支持的指令集包含{ addu, subu, ori, lw, sw, beq, lui, jal, jr,,j nop}。为了实现这些功能，CPU 主要包含了 Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件。

（二）关键模块定义

1. GRF

（可使用表格进行端口说明）

信号名	方向	位数	描述
WPC	I	32	指令的储存地址
clk	I	1	时钟信号
reset	I	1	复位信号
we	I	1	可写入信号
RA1	I	5	读出地址 1
RA2	I	5	读出地址 2
WA	I	5	写入地址
WD	I	32	写入数据
RD1	O	32	读出数据 1
RD2	O	32	读出数据 2

2. DM

信号名	方向	位数	描述

WPC	I	32	指令的储存地址
clk	I	1	时钟信号
reset	I	1	复位信号
MemWrite	I	1	可写入信号
Address	I	32	地址
WD	I	32	写入数据
RD	O	32	读出数据

3. ALU

信号名	方向	描述
In1	I	运算数 1
In2	I	运算数 2
OP	I	操作信号
Zero	I	运算结果是否为 0
Out	I	运算结果

运算符对应表(四位操信号备用)

0	加
1	减
2	或
3	与
4	将 in2 左移 16 位（用于 lui）

4. IFU

信号名	方向	描述
Clk	I	时钟信号
reset	I	复位信号
NPC	I	下一个 PC 地址

cmd	O	指令

5. EXT

用于拓展信号 16->32

00	无符号拓展
01	有符号拓展
10	加载到高位
11	符号拓展后左移 2 位

信号名	方向	描述
Imm	I	16 位立即数
EOp	I	操作符
ext	O	拓展后 32 位数

（三）控制器 Controller 定义

该模块对指令的 op 和 func 进行解码，输出对应的信号。

信号名	方向	位数	描述
op	I	6	指令的 6 位 op 部分
func	I	6	指令的 6 位 func 部分
RegDst	O	2	寄存器写入选择信号 00：使用 rt 写入 01：使用 rd 写入 10：使用 31 号写入
RegWrite	O	1	寄存器写入信号
ALUSrc	O	1	ALU 运算数选择信号 0：grf 1：imm
branch	O	2	PC 变更选择信号 00：自增 4 01：暂无

			10：j/jal 11：jr
beq	O	1	是否 beq 指令
MemWrite	O	1	内存写入信号
toReg	O	2	写入寄存器信号 00：ALU 结果 01：内存 10：PC+4
extsel	O	2	选择拓展信号
ALU	O	4	ALU 操作信号

指令信号对应表

	RegDst	RegWrite	ALUSrc	branch	beq	MemWrite	toReg	extsel	ALU
addu	01	1	0	00	0	0	00	x	0000
subu	01	1	0	00	0	0	00	x	0001
ori	00	1	1	00	0	0	00	00	0010
lw	00	1	1	00	0	0	01	01	0000
sw	x	0	1	00	0	1	x	01	0000
beq	x	0	0	01	1	0	x	x	0001
lui	00	1	1	00	0	0	00	00	0100
j	x	0	x	10	0	0	x	x	x
jal	10	1	x	10	0	0	10	x	x
jr	x	0	x	11	0	0	x	x	x

二、测试方案

（一）典型测试样例

1. ALU 功能测试

```
1
2  ori $t1,3
3  ori $t2,2
4  addu $t3,$t1,$t2
5  subu $t4,$t1,$t2
6  lui $t5,1
```

期望值：

$t1	9	3
$t2	10	2
$t3	11	5
$t4	12	1
$t5	13	65536

实际值：

```
@00003000: $ 9 <= 00000003
@00003004: $10 <= 00000002
@00003008: $11 <= 00000005
@0000300c: $12 <= 00000001
@00003010: $13 <= 00010000
```

2. DM 功能测试

```
1
2  ori $t1,3
3  ori $t2,2
4  sw $t1,0($0)
5  sw $t2,4($0)
6  lw $t3,4($0)
7  lw $t4,0($0)
```

期望值：

$t1	9	3
$t2	10	2
$t3	11	2
$t4	12	3

Address	Value (+0)	Value (+4)
0x00000000	3	2

实际值：

```
@00003000: $ 9 <= 00000003
@00003004: $10 <= 00000002
@00003008: *00000000 <= 00000003
@0000300c: *00000004 <= 00000002
@00003010: $11 <= 00000002
@00003014: $12 <= 00000003
```

3. 其他指令综合测试

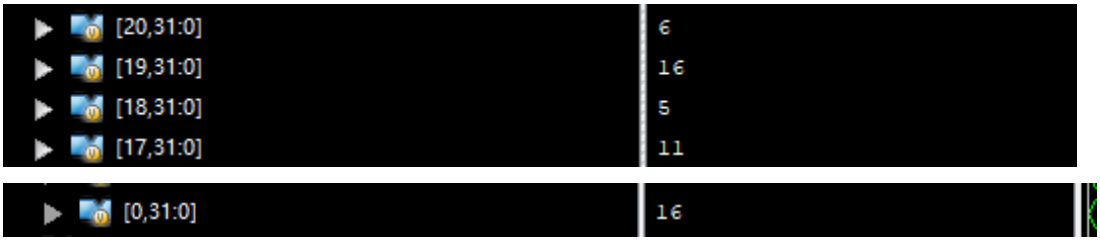
```
ori $s1,$s1,11
ori $s2,$s2,5
addu $s3,$s1,$s2
j this2
this1:
nop
this2:
nop
subu $s4,$s1,$s2
lui $s5,8
sw $s3,0($0)
beq $t1,$t2,that
lw $s6,4($0)
lw $s3,4($0)
that:
lw $s6,4($0)
beq $s1,$s2,this1
```

期望值：

$s1	17	11
$s2	18	5
$s3	19	16
$s4	20	6
$s5	21	524288

	Value (+0)
0x00000000	16

实际值：



@00003000: $17 <= 0000000b

@00003004: $18 <= 00000005

@00003008: $19 <= 00000010

@00003018: $20 <= 00000006

@0000301c: $21 <= 00080000

@00003020: *00000000 <= 00000010

@00003030: $22 <= 00000000

（二）自动测试工具

1. 测试样例生成器

```
import os

import random


with open("D:\\Study\\CO\\P4\\mips5.asm","w") as file:
    for i in range(10):
        x=random.randint(0,27)
        num=random.randint(0,1000)
        file.write("lui $%d,%d\n"%(x,num))
    for i in range(10):
        x=random.randint(0,27)
        y=random.randint(0,27)
        num=random.randint(0,1000)
        file.write("ori $%d,$%d,%d\n"%(x,y,num))
    for i in range(10):
        x=random.randint(0,27)
        y=random.randint(0,27)
```

```
        z=random.randint(0,27)
        file.write("addu $%d,$%d,$%d\n"%(x,y,z))
    for i in range(10):
        x=random.randint(0,27)
        y=random.randint(0,27)
        z=random.randint(0,27)
        file.write("subu $%d,$%d,$%d\n"%(x,y,z))
    # for i in range(5):
    #     x=random.randint(0,27)
    #     y=random.randint(0,27)
    #     num=random.randint(0,10000)
    #     file.write("lw $%d,%d($%d)\n"%(x,4*num,y))
    # for i in range(5):
    #     x=random.randint(0,27)
    #     y=random.randint(0,27)
    #     num=random.randint(0,10000)
    #     file.write("sw $%d,%d($%d)\n"%(x,4*num,y))
    # for i in range(5):
    #     x=random.randint(0,27)
    #     y=random.randint(0,27)
    #     num=random.randint(0,100)
    #     file.write("beq $%d,$%d,%d\n"%(x,y,4*num))

    os.system("D:\\Study\\CO\\P4\\create_test.bat")
```

2. 自动执行脚本

```
java -
jar D:\Software\Mars\mars4_5.jar mips5.asm nc mc C
ompactDataAtZero a dump .text HexText mips\code.tx
t
```

```
java -
jar D:\Software\Mars\mars4_5.jar mips5.asm nc mc C
ompactDataAtZero dump .data HexText out1.txt
    java -
jar D:\Software\Mars\mars4_5.jar mips5.asm nc mc C
ompactDataAtZero $0 $1 $2 $3 $4 $5 $6 $7 $8 $9 $10
 $11 $12 $13 $14 $15 $16 $17 $18 $19 $20 $21 $22 $
23 $24 $25 $26 $27 $28 $29 $30 $31 >out2.txt
```

三、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

因为 mips 中以字节为最小单位，计算出的内存地址的低 2 位代表着一个字中的 4 个字节，而 DM 中以字为一个划分，应该忽略低 2 位，以 2~11 位来代表地址。addr 信号来自于 ALU 计算得到的 32 位结果中的 2~11 位。

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。

1) Always case

```
    always @(*) begin
```

```
        case(op)
            6'b000000:
            //addu,subu
            begin
                RegDst<=2'b01;
                RegWrite<=1;
                ALUSrc<=0;
                MemWrite<=0;
                toReg<=0;
                beq<=0;
                extsel<=0;
                case (func)
                    6'b100001://addu
                    begin
                        ALU <= 0;
                        branch<=2'b00;
                    end
                    6'b100011://subu
                    begin
                        ALU <= 1;
                        branch<=2'b00;
                    end
                    6'b001000://jr
                    begin
                        ALU <= 0;
                        branch<=2'b11;
                    end
                endcase
                 end
```

2) Assign

```
assign addu = (~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(~func[1])&(func[0]);

assign subu = (~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(func[5])&(~func[4])&(~func[3])&(~func[2])&(func[1])&(func[0]);

assign jr = (~op[5])&(~op[4])&(~op[3])&(~op[2])&(~op[1])&(~op[0])&(~func[5])&(~func[4])&(func[3])&(~func[2])&(~func[1])&(~func[0]);

assign ori = (~op[5])&(~op[4])&(op[3])&(op[2])&(~op[1])&(op[0]);
```

```
assign lw = (op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(op[0]);

assign sw = (op[5])&(~op[4])&(op[3])&(~op[2])&(op[1])&(op[0]);

assign beq = (~op[5])&(~op[4])&(~op[3])&(op[2])&(~op[1])&(~op[0]);

assign lui = (~op[5])&(~op[4])&(op[3])&(op[2])&(op[1])&(op[0]);

assign j = (~op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(~op[0]);

assign jal = (~op[5])&(~op[4])&(~op[3])&(~op[2])&(op[1])&(op[0]);



assign RegDst={jal,addu|subu};

assign RegWrite=addu|subu|ori|lw|lui|jal;

assign ALUSrc=ori|lw|sw|lui;

assign branch={j|jal|jr,beq|jr};

assign isbeq=beq;

assign MemWrite=sw;

assign toReg={jal,lw};

assign extsel={1'b0,lw|sw};

          assign ALU ={1'b0,lui,ori,subu|beq};
```

对比：第一种符合人的直观感受，但是语句较长，并且特殊情况需要特判。第二种直接翻译 logisim 中的控制器译码方法，更符合电路的直接接线，但是不直观，需要列出真值表。

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？

都回到初始状态，PC 回到 0x00003000，DM 和 GRF 清空。

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 。

addi 和 add 比 addiu 和 addu 多一个表示是否溢出的信号，若忽略溢出，则二者执行结果没有区别。

5. 根据自己的设计说明单周期处理器的优缺点。

优点：易于设计，每个指令对应一条通路，只需添加多路选择器和控制器信号即可实现任意指令功能

缺点：一个时钟周期执行一条指令，则时钟周期需要取决于运行时间最长的指令，会导致整体效率底下。