单周期 32 位 CPU——Verilog 实现

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，支持的指令集包含{addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}。为了实现这些功能，CPU 主要包含了 Ifu, InstrDecd, Controller, Grf, Alu, Dm, Ext, Sels。

（二）关键模块定义

1. IFU

信号名	方向	描述
clk	I	时钟
reset	I	重置
pcIn	I	32 位下一个指令的地址
pcAdd4	O	32 位当前指令地址+4
pc	O	32 位当前指令地址
instr	O	32 位当前指令值

2. GRF

信号名	方向	描述
clk	I	时钟
reset	I	重置
WE	I	写入使能
RA1	I	5 位读数地址 1
RA2	I	5 位读数地址 2
WA	I	5 位写入的地址
WD	I	32 位写入的数据

RD1	O	32 位地址 1 的数据
RD2	O	32 位地址 2 的数据

3. ALU

信号名	方向	描述
A	I	32 位输入数据 1
B	I	32 位输入数据 2
ctrl	I	5 位控制信号
out	O	32 位输出数据
equal	O	有效则 A、B 相等
bigger	O	有效则 A 大于 B
smaller	O	有效则 A 小于 B

ctrl：0:A+B, 1:A-B, 4:A&B, 5:A|B, 6:B 加载到高位, 7: A+(B 符号扩展)

4. DM

信号名	方向	描述
clk	I	时钟
addr	I	5 位读写地址
wd	I	32 位要写入的数据
rd	O	32 位要读出的数据
re	I	读出使能
we	I	写入使能

5.EXT

把 16 位输入信号 0 扩展到 32 位。

6.Controller

信号名	方向	描述
Low6	I	指令最低 6 位
Op	I	指令最高 6 位
ifWrGrf	O	Grf 使能
ifWrRt	O	有效时 grf 写入地址为 rt，否则为 rd
ifImmExt	O	是否需要 0 扩展立即数
ifReDm	O	Dm 读数使能
ifWrDm	O	Dm 写入使能
ifBeq	O	是否是 beq
ifJal	O	是否是 jal
ifJr	O	是否是 jr

用最高和最低 6 位控制各个指令的实现，再根据指令选择各个控制信号

（三）重要机制实现方法

1.GrfWa 的选择

```
out = (ifJal)? 5'b11111:
      (ifWrRt)? rt:
      rd;
```

2.GrfWd 的选择

```
out = (ifJal)? pcAdd4:
      (ifReDm)? dmOut:
      aluOut;
```

3.AluB 的选择

```
out = (ifImmExt)? immExt:
      grfRD2;
```

4.IfuIn 的选择

```
out = (ifBeq && ifEqual)? (pcAdd4 + ( ((immExt[15] == 0)? (immExt) :
```

(immExt + 32'hffff0000)) << 2)):

(ifJal)? ({pc[31:28], jalTo, 2'b00}):

(ifJr)? (grfRD1):

pcAdd4;

二、测试方案

（一）典型测试样例

1. ALU 功能测试

测试 alu_ctrl 为 0,1,4,5,6,7 的输出

2. Controller 功能测试

测试 Controller 表格中的 low6 和 op 的所有信号，检查输出信号

（二）自动测试

mars 指令：

subu $28 $28 $28

subu $29 $29 $29

ori $0, $0, 0

ori $1, $1, 1

ori $2, $2, 2

ori $3, $3, 3

ori $4, $4, 4

ori $5, $5, 5

ori $6, $6, 6

ori $7, $7, 7

ori $8, $8, 8

ori $9, $9, 9

ori $10, $10, 10

ori $11, $11, 11

```
ori $12, $12, 12
ori $13, $13, 13
ori $14, $14, 14
ori $15, $15, 15
ori $16, $16, 16
ori $17, $17, 17
ori $18, $18, 18
ori $19, $19, 19
ori $20, $20, 20
ori $21, $21, 21
ori $22, $22, 22
ori $23, $23, 23
ori $24, $24, 24
ori $25, $25, 25
ori $26, $26, 26
ori $27, $27, 27
ori $28, $28, 28
ori $29, $29, 29
ori $30, $30, 30
ori $31, $31, 31
ori $1, $1, 1
sw $1, 4($0)
sw $1, 8($0)
sw $1, 12($0)
sw $1, 16($0)
sw $1, 20($0)
sw $1, 24($0)
sw $1, 28($0)
sw $1, 32($0)
sw $1, 36($0)
```

```
sw $1, 40($0)
sw $1, 44($0)
sw $1, 48($0)
sw $1, 52($0)
sw $1, 56($0)
sw $1, 60($0)
sw $1, 64($0)
sw $1, 68($0)
sw $1, 72($0)
sw $1, 76($0)
sw $1, 80($0)
sw $1, 84($0)
sw $1, 88($0)
sw $1, 92($0)
sw $1, 96($0)
sw $1, 100($0)
sw $1, 104($0)
sw $1, 108($0)
sw $1, 112($0)
sw $1, 116($0)
sw $1, 120($0)
sw $1, 124($0)
ori $1, $1, 1
sw $1, 0($0)
lw $2, 0($0)
lw $3, 0($0)
lw $4, 0($0)
lw $5, 0($0)
lw $6, 0($0)
lw $7, 0($0)
```

```
lw $8, 0($0)
lw $9, 0($0)
lw $10, 0($0)
lw $11, 0($0)
lw $12, 0($0)
lw $13, 0($0)
lw $14, 0($0)
lw $15, 0($0)
lw $16, 0($0)
lw $17, 0($0)
lw $18, 0($0)
lw $19, 0($0)
lw $20, 0($0)
lw $21, 0($0)
lw $22, 0($0)
lw $23, 0($0)
lw $24, 0($0)
lw $25, 0($0)
lw $26, 0($0)
lw $27, 0($0)
lw $28, 0($0)
lw $29, 0($0)
lw $30, 0($0)
lw $31, 0($0)
ori $1 $1 907
sw $0 0($0)
lw $1 0($0)
sw $1 4($0)
lw $2 4($0)
sw $2 8($0)
```

```
lw $3 8($0)
sw $3 12($0)
lw $4 12($0)
sw $4 16($0)
lw $5 16($0)
sw $5 20($0)
lw $6 20($0)
sw $6 24($0)
lw $7 24($0)
sw $7 28($0)
lw $8 28($0)
sw $8 32($0)
lw $9 32($0)
sw $9 36($0)
lw $10 36($0)
sw $10 40($0)
lw $11 40($0)
sw $11 44($0)
lw $12 44($0)
sw $12 48($0)
lw $13 48($0)
sw $13 52($0)
lw $14 52($0)
sw $14 56($0)
lw $15 56($0)
sw $15 60($0)
lw $16 60($0)
sw $16 64($0)
lw $17 64($0)
sw $17 68($0)
```

lw $18 68($0)

sw $18 72($0)

lw $19 72($0)

sw $19 76($0)

lw $20 76($0)

sw $20 80($0)

lw $21 80($0)

sw $21 84($0)

lw $22 84($0)

sw $22 88($0)

lw $23 88($0)

sw $23 92($0)

lw $24 92($0)

sw $24 96($0)

lw $25 96($0)

sw $25 100($0)

lw $26 100($0)

sw $26 104($0)

lw $27 104($0)

sw $27 108($0)

lw $28 108($0)

sw $28 112($0)

lw $29 112($0)

sw $29 116($0)

lw $30 116($0)

sw $30 120($0)

lw $31 120($0)

sw $31 124($0)

lui $1 234

lui $2 234

```
lui $3 234
lui $4 234
lui $5 234
lui $6 234
lui $7 234
lui $8 234
lui $9 234
lui $10 234
lui $11 234
lui $12 234
lui $13 234
lui $14 234
lui $15 234
jal con
lui $1 222
subu $16 $16 $1
subu $17 $17 $1
subu $18 $18 $1
subu $19 $19 $1
subu $20 $20 $1
subu $21 $21 $1
subu $22 $22 $1
subu $23 $23 $1
subu $24 $24 $1
subu $25 $25 $1
subu $26 $26 $1
subu $27 $27 $1
subu $28 $28 $1
subu $29 $29 $1
subu $30 $30 $1
```

```
jal end
con:
addu $16 $16 $1
addu $17 $17 $2
addu $18 $18 $3
addu $19 $19 $4
addu $20 $20 $5
addu $21 $21 $6
addu $22 $22 $7
addu $23 $23 $8
addu $24 $24 $9
addu $25 $25 $10
addu $26 $26 $11
addu $27 $27 $12
addu $28 $28 $13
addu $29 $29 $14
addu $30 $30 $15
jr $31
end:
```

三、思考题

（一）

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？

默认地址是 4 的倍数，所以最后两位为 0，算地址的时候可以省略不计，addr 是 alu 的输出。

（二）

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。

先经过与门获得指令，在经过或门获得控制信号，比如：

```
assign addu = ((op == 6'b000000) && (low6 == 6'b100001));
assign subu = ((op == 6'b000000) && (low6 == 6'b100011));
assign ori = (op == 6'b001101);
assign lui = (op == 6'b001111);
assign lw = (op == 6'b100011);
assign sw = (op == 6'b101011);
assign beq = (op == 6'b000100);
assign jal = (op == 6'b000011);
assign jr = ((op == 6'b000000) && (low6 == 6'b001000));

assign ifWrGrf = addu | subu | ori | lui | lw | jal;
assign ifWrRt = ori | lui | lw;
assign ifImmExt = ori | lui | lw | sw;
assign ifReDm = lw;
assign ifWrDm = sw;
assign ifBeq = beq;
assign ifJal = jal;
assign ifJr = jr;
```

如果直接从 32 位指令到控制信号，容易混乱。先获得指令名称，再获得控制信号更有条理，但是定义的连线会增多。

（三）

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？

都由时钟信号驱动，因此在判断时要先考虑 reset。

（四）

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 是加立即数指令，支持溢出检测，具体表现为：遇到溢出时，溢出错误标志变为高电平，传送到控制器 cu 中，导致此时的寄存器写使能信号 regwr 无效，最终结果为不将运算结果写入目的寄存器。

addiu 是加立即数指令，不受溢出限制，具体表现为：遇到溢出时，对溢出的结果进行 32bit 求模，将求模结果写入目的寄存器中，因而不受溢出限制。

在忽略溢出的前提下，addi 和 addiu 都能计算出溢出后的结果（进行 32bit 求模），并且将结果写入目的寄存器中，因此二者是等价的。

（五）

根据自己的设计说明单周期处理器的优缺点。

优点在于设计简单，清晰明了；缺点在于不同指令所需要的时间不同，但是不能共用某些元件（如运算器），使用单周期要用更多的运算器。