计算机组成原理实验报告

18375200 刘裕炜

一、CPU设计方案综述

（一）总体设计概述

本CPU为Logisim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、GRF、ALU、DM、控制器等模块。

（二）关键模块定义

1、GRF

表1 GRF模块接口

信号名	方向	描述
RAddr0[4:0]	I	要读取寄存器的第一个地址
RAddr1[4:0]	I	要读取寄存器的第二个地址
WAddr[4:0]	I	写入地址
WriteData[31:0]	I	写入数据
RegWrite	I	写入使能
clk	I	时钟
reset	I	同步复位信号
RData0[31:0]	O	第一个地址寄存器的值
RData1[31:0]	O	第二个地址寄存器的值

表2 GRF功能定义

序号	功能名称	功能描述
1	读寄存器	RData0输出RAddr0地址所寻址的寄存器，RData1输出RAddr1地址所寻址的寄存器
2	写寄存器	RegWrite有效且reset不为1时，在时钟上升沿将WD的数据写入A3地址所寻址的寄存器
3	同步复位	clk上升沿时若reset信号有效，将32个GPR单元清零

2、IM

表3 IM模块接口

信号名	方向	描述
OpAddr[31:0]	I	指令地址
OpCode[31:0]	O	指令数

3、IFU

表4 IFU模块接口

信号名	方向	描述
NPC[31:0]	I	下一条指令地址
reset	I	异步复位信号
clk	I	时钟
PC[31:0]	O	当前指令地址
Instr[31:0]	O	指令
OpCode[5:0]	O	操作码
rs[4:0]	O	rs字段
rt[4:0]	O	rt字段
rd[4:0]	O	rd字段
Shamt[4:0]	O	Shamt字段
Func[5:0]	O	Func字段
imm16[15:0]	O	16位立即数字段
imm26[25:0]	O	26位立即数字段

IFU功能是从 IM 中取指并将指令按字段输出，并向 PC 更新下一条指令的值。

4、ALU

表5 ALU模块接口

信号名	方向	描述
In0[31:0]	I	第一个运算数
In1[31:0]	I	第二个运算数
ALUOp[1:0]	I	ALU控制信号
Zero	O	两数是否相等
Res[31:0]	O	运算结果

表6 ALU功能定义

序号	功能名称	功能描述
1	加法	Res输出In0+In1的值
2	减法	Res输出In0-In1的值
3	按位或	Res输出In0和In1按位或的值
4	判断相等	若In0和In1相等，Zero输出1，否则输出0

5、DM

表7 DM模块接口

信号名	方向	描述
Addr[31:0]	I	操作地址
WData[31:0]	I	写入数据
MemWrite	I	写入使能
clk	I	时钟
reset	I	异步复位信号
RData[31:0]	O	操作地址对应内存单元存储的值

6、NPC

表8 NPC模块接口

信号名	方向	描述
PC[31:0]	I	当前PC值
imm32[31:0]	I	32位立即数
imm26[25:0]	I	26位立即数
rsData[31:0]	I	rs寄存器的值
branch	I	branch信号
jump	I	jump信号
	1	jr信号
NextPC[31:0]	O	下一条PC值
PC_4[31:0]	O	PC+4

7、控制器

控制器由与逻辑和或逻辑两部分组成。与逻辑模块负责将指令进行译码，确定指令类型，并将指令类型传输给或逻辑模块。或逻辑模块负责根据指令类型激活相应的控制信号。以下为或逻辑模块的控制信号真值表。

表9 控制信号真值表

	RegWrite	ALUOp[1:0]	Branch	Jump	Jr	RegDst[1:0]	ALUSel	MemWrite	RegWriteSel[2:0]	Sign	LShift
addu	1	00	0	0	0	01	0	0	000	x	x
subu	1	01	0	0	0	01	0	0	000	x	x
ori	1	10	0	0	0	00	1	0	000	0	x
lw	1	00	0	0	0	00	1	0	001	1	x
sw	0	00	0	0	0	x	1	1	x	1	x
beq	0	01	1	0	0	x	0	0	x	x	x
lui	1	x	0	0	0	00	x	0	010	x	0
jal	1	x	0	1	0	10	x	0	011	x	x
jr	0	x	0	0	1	x	x	0	x	x	x
sll	1	x	0	0	0	01	x	0	010	x	1

二、测试方案

测试1（Fibonacci数列生成）

```
.data
number: .space 400
.text
addu $s0, $0, $0
ori $s1, $0, 45
ori $s2, $0, 0
ori $s3, $0, 4
ori $s4, $0, 1
ori $t0, $0, 0
ori $t1, $0, 1
Loop_Start:
    beq $s0, $s1, Loop_End
    addu $t2, $t1, $t0
    sw $t2, 0($s2)
    addu $s2, $s2, $s3
    addu $s0, $s0, $s4
    subu $t0, $t1, $zero
    addu $t1, $t2, $0
    jal Loop_Start
Loop_End:
lw $t7, -20($s2)
```

测试2

python自动生成，手工做修改以检测jal和beq执行情况。

```
lui $0, 0x3000
lui $1, 0x3001
lui $2, 0x3002
lui $3, 0x3003
lui $4, 0x3004
lui $5, 0x3005
```

```
lui $6, 0x3006
lui $7, 0x3007
lui $8, 0x3008
lui $9, 0x3009
lui $10, 0x300a
lui $11, 0x300b
lui $12, 0x300c
lui $13, 0x300d
lui $14, 0x300e
lui $15, 0x300f
lui $16, 0x3010
lui $17, 0x3011
lui $18, 0x3012
lui $19, 0x3013
lui $20, 0x3014
lui $21, 0x3015
lui $22, 0x3016
lui $23, 0x3017
lui $24, 0x3018
lui $25, 0x3019
lui $26, 0x301a
lui $27, 0x301b
lui $28, 0x301c
lui $29, 0x301d
lui $30, 0x301e
lui $31, 0x301f
ori $0, $0, 0x1
ori $1, $0, 0x2
ori $2, $0, 0x3
ori $3, $0, 0x4
ori $4, $0, 0x5
ori $5, $0, 0x6
ori $6, $0, 0x7
ori $7, $0, 0x8
ori $8, $0, 0x9
ori $9, $0, 0xa
ori $10, $0, 0xb
ori $11, $0, 0xc
ori $12, $0, 0xd
ori $13, $0, 0xe
ori $14, $0, 0xf
ori $15, $0, 0x10
ori $16, $0, 0x11
ori $17, $0, 0x12
ori $18, $0, 0x13
ori $19, $0, 0x14
ori $20, $0, 0x15
ori $21, $0, 0x16
ori $22, $0, 0x17
ori $23, $0, 0x18
ori $24, $0, 0x19
ori $25, $0, 0x1a
ori $26, $0, 0x1b
ori $27, $0, 0x1c
ori $28, $0, 0x1d
ori $29, $0, 0x1e
ori $30, $0, 0x1f
ori $31, $0, 0x20
```

```
addu $31, $31, $31
addu $30, $30, $30
addu $29, $29, $29
addu $28, $28, $28
addu $27, $27, $27
addu $26, $26, $26
addu $25, $25, $25
addu $24, $24, $24
addu $23, $23, $23
addu $22, $22, $22
addu $21, $21, $21
addu $20, $20, $20
addu $19, $19, $19
addu $18, $18, $18
addu $17, $17, $17
addu $16, $16, $16
addu $15, $15, $15
addu $14, $14, $14
addu $13, $13, $13
addu $12, $12, $12
addu $11, $11, $11
addu $10, $10, $10
addu $9, $9, $9
addu $8, $8, $8
addu $7, $7, $7
addu $6, $6, $6
addu $5, $5, $5
addu $4, $4, $4
addu $3, $3, $3
addu $2, $2, $2
addu $1, $1, $1
addu $0, $0, $0
subu $0, $0, $15
subu $1, $1, $15
subu $2, $2, $15
subu $3, $3, $15
subu $4, $4, $15
subu $5, $5, $15
subu $6, $6, $15
subu $7, $7, $15
subu $8, $8, $15
subu $9, $9, $15
subu $10, $10, $15
subu $11, $11, $15
subu $12, $12, $15
subu $13, $13, $15
subu $14, $14, $15
subu $15, $15, $15
subu $16, $16, $15
subu $17, $17, $15
subu $18, $18, $15
subu $19, $19, $15
subu $20, $20, $15
subu $21, $21, $15
subu $22, $22, $15
subu $23, $23, $15
subu $24, $24, $15
subu $25, $25, $15
```

```
subu $26, $26, $15
subu $27, $27, $15
subu $28, $28, $15
subu $29, $29, $15
subu $30, $30, $15
subu $31, $31, $15
ori $5, $0, 4
sw $0, 0($5)
sw $1, 4($5)
sw $2, 8($5)
sw $3, 12($5)
sw $4, 16($5)
sw $5, 20($5)
sw $6, 24($5)
sw $7, 28($5)
sw $8, 32($5)
sw $9, 36($5)
sw $10, 40($5)
sw $11, 44($5)
sw $12, 48($5)
sw $13, 52($5)
sw $14, 56($5)
sw $15, 60($5)
sw $16, 64($5)
sw $17, 68($5)
sw $18, 72($5)
sw $19, 76($5)
sw $20, 80($5)
sw $21, 84($5)
sw $22, 88($5)
sw $23, 92($5)
sw $24, 96($5)
sw $25, 100($5)
sw $26, 104($5)
sw $27, 108($5)
sw $28, 112($5)
sw $29, 116($5)
sw $30, 120($5)
sw $31, 124($5)
lw $0, 128($zero)
lw $1, 124($zero)
lw $2, 120($zero)
lw $3, 116($zero)
lw $4, 112($zero)
lw $5, 108($zero)
lw $6, 104($zero)
lw $7, 100($zero)
lw $8, 96($zero)
lw $9, 92($zero)
lw $10, 88($zero)
lw $11, 84($zero)
lw $12, 80($zero)
lw $13, 76($zero)
lw $14, 72($zero)
lw $15, 68($zero)
lw $16, 64($zero)
lw $17, 60($zero)
lw $18, 56($zero)
```

```
lw $19, 52($zero)
lw $20, 48($zero)
lw $21, 44($zero)
lw $22, 40($zero)
lw $23, 36($zero)
lw $24, 32($zero)
lw $25, 28($zero)
lw $26, 24($zero)
lw $27, 20($zero)
lw $28, 16($zero)
lw $29, 12($zero)
lw $30, 8($zero)
lw $31, 4($zero)
ori $15, $zero, 0x0004
ori $16, $zero, 0x0100
ori $17, $zero, 0x0000
Loop0:
    beq $16, $0, EndLoop
    jal Loop1
Loop1:
    jal Loop2
Loop2:
    jal Loop3
Loop3:
    jal Loop4
Loop4:
    jal Loop5
Loop5:
    jal Loop6
Loop6:
    jal Loop7
Loop7:
    jal Loop8
Loop8:
    jal Loop9
Loop9:
    jal Loop10
Loop10:
    subu $16, $16, $15
    jal Loop0
EndLoop:
nop
```

测试3

对jr进行测试，包括正常的$ra寄存器和一般的$16寄存器。

```
ori $24, $zero, 0x3000  #text base addr
ori $31, $0, 22  #jump to NopEnd
sll $31, $31, 2
addu $31, $31, $24
ori $2, $0, 4
ori $3, $0, 0x100
ori $4, $0, 1
ori $5, $5, 0
Loop:
    beq $3, $zero, EndLoop
```

```
    ori $16, $0, 17  #jump to jr_test2
    sll $16, $16, 2
    addu $16, $16, $24
    addu $5, $5, $4
    subu $3, $3, $2
    jal Loop
jr_test1:
    addu $9, $9, $4
    jr $16
jr_test2:
    addu $10, $10, $4
    jr $ra
EndLoop:
    addu $7, $7, $4
    jal jr_test1
addu $8, $8, $4
NopEnd:
nop
```

三、思考题

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?

位数是[11:2]是因为DM是1K×32的存储器，其每个存储单元是32位，即4字节，因此要求传入地址是4字节对齐的，所以最终寻址的芯片地址是addr地址除以4，即右移两位。addr信号即是从ALU计算得出的读/写地址。

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。

可以直接译码，如

```
assign Branch = $ {!OpCode[5:3], OpCode[2], !OpCode[1:0]};
```

也可以采用与逻辑和或逻辑两步走，先将操作码译为指令类型，再将指令类型译为控制信号，如

```
// And Logic
assign beq = $ {!OpCode[5:3], OpCode[2], !OpCode[1:0]};
//......

//Or Logic
assign Branch = | {beq};
//......
```

前者优点是节省元件，但缺点是难以调试、修改、添加新指令支持；

后者优点是可读性好，易于修改、添加新指令，缺点是需要多一个元件。

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?

reset所驱动的部件（PC，GRF，DM）都为时序逻辑部件，即都是具有存储功能的部件。

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。

add和addi使用双符号位来判断溢出与否，双符号位对低32位的运算没有影响，而C语言会忽略溢出的提示，只取最终运算结果，因此addi与addiu等价，add与addu等价。

根据自己的设计说明单周期处理器的优缺点。

单周期处理器的优点在于设计较为简单，不需要考虑时序问题。缺点在于其时钟周期受关键路径影响，且同时只能进行一条指令的执行，因此效率较为低下，且时钟频率不能太高，运算速度较慢。