P4 Verilog 单周期 CPU 设计文档

19373490-吴昊天

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 Verilog 设计的单周期处理器，支持的指令集为{addu,subu,ori,sw,lw,beq,lui,sltu,jal,jr,nop}。整体设计中主要包括 IFU，GRF，ALU，DM，Controller 以及 MUX 若干。顶层有效信号为时钟信号 clk 和复位信号 reset，复位为同步复位。

（二）关键模块定义

1、IFU（取指令单元）：内部包括 PC（程序计数器）、NPC（次地址计算单元）、IM（指令存储器）等相关逻辑。

（1）PC 用 32 位寄存器实现，具有同步复位功能，复位至 0x0000_3000.

表 1、PC 描述

功能描述	32 位可复位寄存器，reset 高电平有效。 同步复位至 0x0000_3000	
信号名	方向	描述
Clk	I	MIPS-C 处理器时钟
Reset	I	复位信号，同步复位
NPC[31:0]	I	32 位次地址输入
PC[32:0]	O	32 位输出，当前地址

（2）IM 用寄存器阵列实现，大小为 32bit×1024 字。初始读取指令时以系统任务$readmemh 读取 code.txt 的十六进制机器码。

表 2、IM 描述

功能描述	寄存器阵列，存储指令。以$readmemh 读取 code.txt 大小为 32×1024 字	
信号名	方向	描述
addr[31:0]	I	取值地址，取 addr[11:2]

Instr[31:0]	O	32 为读出的指令

（3）NPC 相当于一个组合逻辑，输入 PC 以及相关辅助信号，根据控制信号 NPCOp 选择计算方式，输出次地址 NPC。

表 3、NPC 描述

信号名	方向	描述
PC[31:0]	I	当前地址
Instr[31:0]	I	32 位指令
NPCOp[2:0]	I	NPC 地址选择： 000：PC+4（计算顺序地址） 001：计算 beq 地址 010：计算 jal 地址 011：计算 jr 地址
zero	I	rs 和 rt 相等判断 1：相等 0：不相等
RA[31:0]	I	32 位输入地址
NPC[31:0]	O	32 位次地址
PC4[31:0]	O	32 位 PC+4

2、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）。具有写使能的寄存器阵列，大小为 32bit×32 字，具有同步复位功能。

表 4、GRF 描述

功能描述	寄存器组，reset 同步复位	
信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号，同步复位
RFWr	I	使能信号，时钟上升沿 时为 1 有效
A1[4:0]	I	第一个读出寄存器编号

A2[4:0]	I	第二个读出寄存器编号
A3[4:0]	I	回写寄存器编号
WData[31:0]	I	回写寄存器的数据值
WPC[31:0]	I	应题目输出要求，当前执行指令
RD1[31:0]	O	第一个读出的寄存器值
RD2[31:0]	O	第二个读出的寄存器值

3、ALU（算术逻辑单元）。提供 32 位加、减、或运算及比较大小功能，控制信号为 ALUOp。

表 5、ALU 描述

信号名	方向	描述
A[31:0]	I	第一个操作数
B[31:0]	I	第二个操作数
ALUOp[2:0]	I	ALU 功能选择 000：A+B 001：A-B 010：A or B 011：SLT 其余保留
C[31:0]	O	32 位计算结果
Zero	O	A 和 B 相等比较结果 1：A＝B 0：A≠B

4、DM（数据存储器）由寄存器阵列表示。Reset 同步复位，写使能 DMWr 为 1 时可以写入。

表 6、DM 描述

功能描述	寄存器阵列，存储数据。Reset 同步复位 以$display 按要求输出，大小为 32×1024 字

信号名	方向	描述
Clk	I	时钟信号
Reset	I	同步复位信号
addr[31:0]	I	访问地址，取 addr[11:2]
pc[31:0]	I	按要求输出当前 PC 地址
DMWr	I	DM 写使能信号
MemWrite[31:0]	I	DM 写入数据
MemRead[31:0]	I	DM 读出数据

5、EXT（扩展单元），控制信号为 EXTOp 以选择功能。

表 7、EXT 描述

信号名	方向	描述
Imm[15:0]	I	16 位立即数
EXTOp[1:0]	I	EXT 功能选择信号 00：符号扩展 01：无符号扩展 10：加载立即数至高位 11：保留
ext[31:0]	O	EXT 输出数据

（三）控制器设计及控制单元

用 Verilog 设计时不必再像 logisim 那样将 Control_and 和 Control_or 分开为两个单另的模块。以 parameter 的方式定义 Rtype=6’b000000，以及其他指令的 opcode 区域或是 funct；每个指令对应定义一个 wire，例如以 assign addu = (op==RType)&&(funct==addu)来对应 addu 指令。输出为各控制信号，以逻辑或的方式表示各控制信号，例如 assign RFWr = addu || subu || ori || lw || lui || jal || sltu 来表示寄存器写使能信号。

在 CPU 中总共有 8 个控制信号，功能区分如下表：

表 8、控制单元描述

信号名	描述

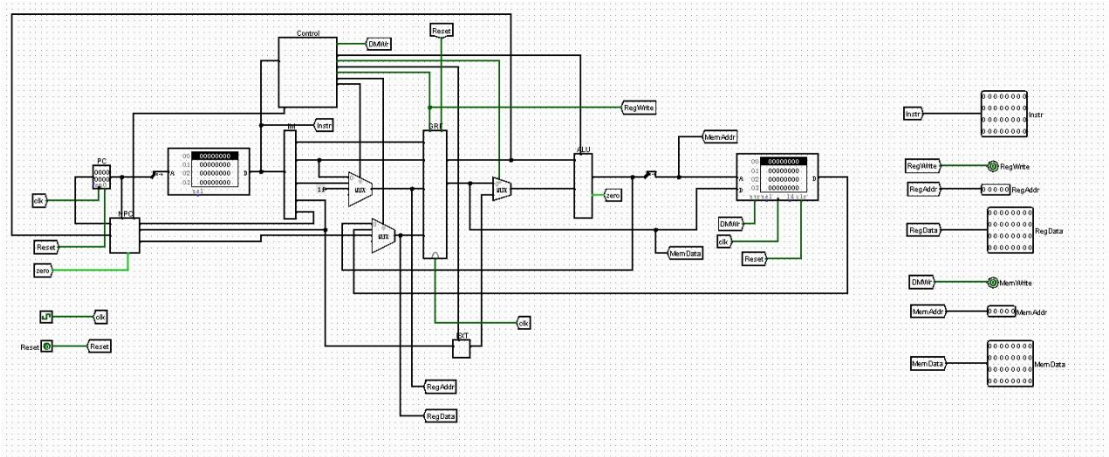
NPCOp[2:0]	NPC 地址选择： 000：PC+4（计算顺序地址） 001：计算 beq 地址 010：计算 jal 地址 011：计算 jr 地址
M1Sel[1:0]	回写寄存器选择：（写哪里） 00：回写 rt 01：回写 rd 10：回写$ra 11：保留
M2Sel[1:0]	回写数据选择：（写什么） 00：ALU 计算所得数据 01：DM 读出数据 10：PC+4 11：保留
RFWr	寄存器写使能信号 1：上升沿写入数据 0：写入无效
EXTOp[1:0]	扩展单元功能选择： 00：符号扩展 01：无符号扩展 10：加载立即数至高位 11：保留
M3Sel	ALU 计算数选择： 0: 寄存器读出数据二 1：扩展单元所得立即数
ALUOp[2:0]	ALU 功能选择 000：A+B 001：A-B

	010：A or B 011：SLT 其余保留
DMWr	DM 写使能信号： 1：上升沿数据写入 0：写入无效

表 9、指令与信号对应

指令\信号名	NPCOp	M1Sel	M2Sel	RFWr	EXTOp	M3Sel	ALUOp	DMWr
addu	000	01	00	1	00	0	000	0
subu	000	01	00	1	00	0	001	0
ori	000	00	00	1	01	1	010	0
lw	000	00	01	1	00	1	000	0
sw	000	00	00	0	00	1	000	1
beq	001	00	00	0	00	0	001	0
lui	000	00	00	1	10	1	000	0
sltu	000	01	00	1	00	0	011	0
jal	010	10	10	1	00	0	000	0
jr	011	00	00	0	00	0	000	0

二、数据通路图示



三、测试方案

（一）测试代码

```
lui $t0,0x1234
lui $t1,0xff45
ori $t0,0x2356
ori $t1,0x6789

addu $t2,$t0,$t1
addu $t3,$t1,$t1
subu $t4,$t1,$t0

sw $t3,0($0)
sw $t4,4($0)
beq $t2,$t3,loop1
lw $a0,0($0)
loop1:
jal loop
beq $t2,$t2,loop2
subu $t7,$t0,$t1

loop:
addu $t5,$t0,$t1
jr $ra
loop2:
lw $a1,4($0)
```

（二）测试结果

预期结果：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0xfe8acf12
$a1	5	0xed114433
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x12342356
$t1	9	0xff456789
$t2	10	0x11798adf
$t3	11	0xfe8acf12
$t4	12	0xed114433
$t5	13	0x11798adf
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003030
pc		0x00003044
hi		0x00000000
lo		0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0xfe8acf12	0xed114433	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000001a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

测试结果：

DM & GRF：

	0	1
0x0	FE8ACF12	ED114433
0x2	00000000	00000000
0x4	00000000	00000000
0x6	00000000	00000000
0x8	00000000	00000000
0xA	00000000	00000000
0xC	00000000	00000000
0xE	00000000	00000000
0x10	00000000	00000000
0x12	00000000	00000000
0x14	00000000	00000000
0x16	00000000	00000000
0x18	00000000	00000000
0x1A	00000000	00000000
0x1C	00000000	00000000
0x1E	00000000	00000000

	0	1
0x0	00000000	00000000
0x2	00000000	00000000
0x4	FE8ACF12	ED114433
0x6	00000000	00000000
0x8	12342356	FF456789
0xA	11798ADF	FE8ACF12
0xC	ED114433	11798ADF
0xE	00000000	00000000
0x10	00000000	00000000
0x12	00000000	00000000
0x14	00000000	00000000
0x16	00000000	00000000
0x18	00000000	00000000
0x1A	00000000	00000000
0x1C	00000000	00000000
0x1E	00000000	00003030

输出结果：

@00003000: $ 8 <= 12340000

@00003004: $ 9 <= ff450000

@00003008: $ 8 <= 12342356

@0000300c: $ 9 <= ff456789

@00003010: $10 <= 11798adf

@00003014: $11 <= fe8acf12

@00003018: $12 <= ed114433

@0000301c: *00000000 <= fe8acf12

@00003020: *00000004 <= ed114433

@00003028: $ 4 <= fe8acf12

@0000302c: $31 <= 00003030

@00003038: $13 <= 11798adf

@00003040: $ 5 <= ed114433

四、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

DM 中每个字长 4 个字节，一个字用一个 32 位的 reg 来表示和存储。因此地址每次加 4，DM 寄存器组下标加 1，所以从第三位开始取。DM 大小为 32bit*1024，所以取[11:2]这 10 位作为 DM 选择 reg 的地址。addr 信号来自 ALU 的计算结果。

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。

Verilog 语言设计控制器可以选择以下几种译码方式：

（1）指令驱动型：

```
case (Instr) begin
    addu : begin
    ****
    end
    subu : begin
    ****
    end
endcase
```

（2）控制信号驱动型：

```
 Wire addu,subu,***;
assign DMWr = addu || subu || ***;
```

优缺点比较：指令驱动型罗列清晰，直观易懂，方便对于新增指令的添加，而且不容易遗漏控制信号。但是整体代码量明显增多，比较繁琐。

控制信号驱动型在指令较多时比较简练，不足之处是如果错添或漏添某条指

令，debug 比较困难，可读性不好。

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？

PC，GRF，DM。PC 需要从指令存储器指定的起始地址开始取值，GRF 和 DM 需要初始化为 0。这些部件都具有记忆存储功能，所以在 CPU 开始使用前需要复位和初始化，否则其中的值不确定 CPU 无法正常工作。

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 。

add 将两个操作数符号扩展一位，再通过相加后扩展位与原最高位是否相同来判断是否溢出。在忽略溢出的前提下，add 肯定不会触发异常，将计算值赋值给目的寄存器的操作与 addu 无异。addi 和 addiu 也类似。

5、根据自己的设计说明单周期处理器的优缺点。

优点：单周期处理器结构比较直观明了，易于理解。所用的部件比流水线少，每条指令一个周期完成，不会产生冲突和冒险，易于维护和扩展。

缺点：运行效率比较慢，时钟周期由执行时间最长的指令的关键路径决定，部分部件在很多时候闲置，没有充分利用。