单周期CPU设计文档

概述

本次项目通过Logisim设计的单周期MIPS架构CPU初步支持了包括add, sub, ori, lw, sw, beq,, lui, nop的指令，CPU子模块包括IFU、NPC、Controller、GRF、ALU、DM、EXT六个部分。

数据通路模块

IFU（取指令单元）

模块内部包含PC（程序计数器）以及IM（指令存储器），分别用Logisim中的32位寄存器和ROM实现（24 * 32 bit）。

端口定义

信号名	方向	位宽	描述
CLK	I	1	时钟信号
Reset	I	1	异步复位信号
Next	I	32	下一条要被执行的指令的地址
Now	O	32	输出当前正在执行的指令的地址
Instr	O	32	输出当前正在执行的指令

功能定义

序号	功能名称	功能描述
1	复位	当 Reset 信号有效时，将 PC 寄存器中的值置为 0x00000000
2	写 PC 寄存器	当 时钟上升沿来临时，将下一条指令的地址（next PC）写入 PC 寄存器
3	取指令	根据当前 PC 的值从 IM（指令存储器）中读出对应的指令到 Instr 端口

GRF（通用寄存器组）

该模块内部包含 32 个具有写使能 32 位寄存器，分别对应 MIPS 架构中0~31通用寄存器（其中 0 号寄存器中的值恒为 0，即不具备写使能）。GRF 可以实现异步复位，同时可以根据输入的 5 位地址（0~31）向寄存器堆存取数据，实现定向访存寄存器。

端口定义

信号名	方向	位宽	描述
CLK	I	1	时钟信号
reset	I	1	异步复位信号
1：复位信号有效			

信号名	方向	位宽	描述
0：复位信号无效			
A1	I	5	地址输入信号，指定 32 个寄存器中的一个，将其中的数据读出到 RD1
A2	I	5	地址输入信号，指定 32 个寄存器中的一个，将其中的数据读出到 RD2
A3	I	5	地址输入信号，指定 32 个寄存器中的一个，将其作为写入目标
WD	I	32	数据输入信号
WE	I	1	写使能信号
1：写入有效			
0：写入失效			
RD1	O	32	输出 A1 指定的寄存器中的 32 位数据
RD2	O	32	输出 A2 指定的寄存器中的 32 位数据

功能定义

序号	功能名称	功能描述
1	复位	Reset 信号有效时，所有寄存器中储存的值均被清零
2	读数据	读出 A1，A2 地址对应的寄存器中储存的数据，将其加载到 RD1 和 RD2
3	写数据	当 WE 信号有效且时钟上升沿来临时，将 WD 中的数据写入到 A3 地址对应的寄存器

NPC

负责计算下一条指令的地址并传递给PC。

信号名	方向	位宽	描述
Now	I	32	当前指令地址
Extender	I	32	地址偏移量
Branch?	I	1	B 类指令选择信号
1：说明当前指令为 B 类指令			
0：说明当前指令不是 B 类信号			
Next	O	32	输出下一指令地址

ALU

该模块主要实现了加法、减法、按位或、立即数加载至高位（LUI）运算。

端口定义

信号名	方向	位宽	描述
ALU_Op	I	3	ALU 功能选择信号
Src2	I	32	参与 ALU 计算的第一个值
Src1	I	32	参与 ALU 计算的第二个值
Shift	I	5	移位数输入
Equal	O	1	相等判断信号
1：Src1 和 Src2 相等			
0：Src1 和 Src2 不相等			
ALU_Result	O	32	输出 ALU 计算结果

功能定义

序号	功能名称	ALU_Op	功能描述
1	减法	0'b000	ALU_Result = Src1 - Src2
2	加法	0'b001	ALU_Result = Src1 + Src2
3	按位或	0'b010	ALU_Result = Src1 | Src2
4	加载至高位	0'b011	ALU_Result = Src2 << 16

DM（数据存储器）

该模块通过Logisim的RAM部件实现，具有读写功能以及异步复位功能

端口定义

信号名	方向	位宽	描述
CLK	I	1	时钟信号
reset	I	1	异步复位信号
A	I	5	地址输入信号，指向数据储存器中某个存储单元
D	I	32	数据输入信号
str	I	1	写使能信号
Data	O	32	输出 A 指定的存储单元中的 32 位数据

Controller（控制模块）

在控制模块中，我们对指令中 Opcode 域和 Funct 域中的数据进行解码，输出 ALUOp,MemtoReg 等8条控制指令，从而对数据通路进行调整，满足不同指令的需求。为实现该模块，我们又在内部设计了两部分 —— 和逻辑（AND Logic）和或逻辑（OR Logic）。前者的功能是识别，将输入的 Opcode 和 Funct 数据识别为对应的指令，后者的功能是生成，根据输入指令的不同产生不同的控制信号。

控制信号

序号	信号名	位宽	描述	触发指令（信号为 1）
1	MemToReg	1	GRF 中 WD 接口输入数据选择	lw
2	MemWrite	1	DM 写入使能信号	sw
3	ALUSrc	1	ALU 中 Src2 接口输入数据选择	ori, lw, sw, lui
4	RegWrite	1	GRF 写入使能信号	add, sub, ori, lw, lui
5	SignedExt	1	立即数符号扩展选择	lw, sw, beq
6	RegDst	1	GRF 中 A3 接口输入数据选择	add, sub
7	Branch	1	B 类指令判断信号	beq
8	ALUCtrl	4	ALU 功能选择信号	

思考题

1：上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

答：状态存储：IFU、GRF、DM；状态转移：NPC、Controller、ALU、EXT。

2：现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用 Register，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

答：我认为是合理的。ROM为只读寄存器，与IM的特性相同；RAM为读写寄存器且带有异步复位功能，与DM特性相同；GRF寄存器堆刚好可以使用Register实现。

3：在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

答：我还设计了NPC模块，将有关PC的计算部分独立出来作为一个模块。

4：事实上，实现 `nop` 空指令，我们并不需要将它加入控制信号真值表，为什么？

答：空指令到来后，控制信号均为假，已实现不进行任何操作的目的，所以不需要额外加入控制信号真值表。

5：阅读 Pre 的 “MIPS 指令集及汇编语言” 一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。

答：我认为测试样例覆盖率较好，但是数据范围较小，且未设计$0寄存器的测试。并且未测试sub指令。

测试方案

基于Pre教程中的测试样例：

```
ori $a0, $0, 123
ori $a1, $a0, 456
lui $a2, 123             # 符号位为 0
lui $a3, 0xffff          # 符号位为 1
ori $a3, $a3, 0xffff     # $a3 = -1
add $s0, $a0, $a2       # 正正
add $s1, $a0, $a3       # 正负
add $s2, $a3, $a3       # 负负
ori $t0, $0, 0x0000
sw $a0, 0($t0)
sw $a1, 4($t0)
sw $a2, 8($t0)
sw $a3, 12($t0)
sw $s0, 16($t0)
sw $s1, 20($t0)
sw $s2, 24($t0)
lw $a0, 0($t0)
lw $a1, 12($t0)
sw $a0, 28($t0)
sw $a1, 32($t0)
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, loop1      # 不相等
beq $a0, $a2, loop2      # 相等
loop1:sw $a0, 36($t0)
loop2:sw $a1, 40($t0)
```

进行了一些补充：

```
ori $a0, $0, 65535
ori $a1, $0, 1234
ori $a1, $a0, 0
lui $t1, 0xffff
ori $t1, $0, 0xffff
ori $t0, 1
add $t2, $t1, $t0
add $t3, $t0, $0
lui $0, 1111
```