P3 Logisim单周期CPU设计文档

1. 整体结构

本次CPU设计为Logisim单周期CPU（32位）设计，基本思路是通过MUX、Splitter等内置器件将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）等基本部件连接成数据通路, 可支持的指令集: **{addu, subu, ori, lw, sw, beq, lui, nop}。**

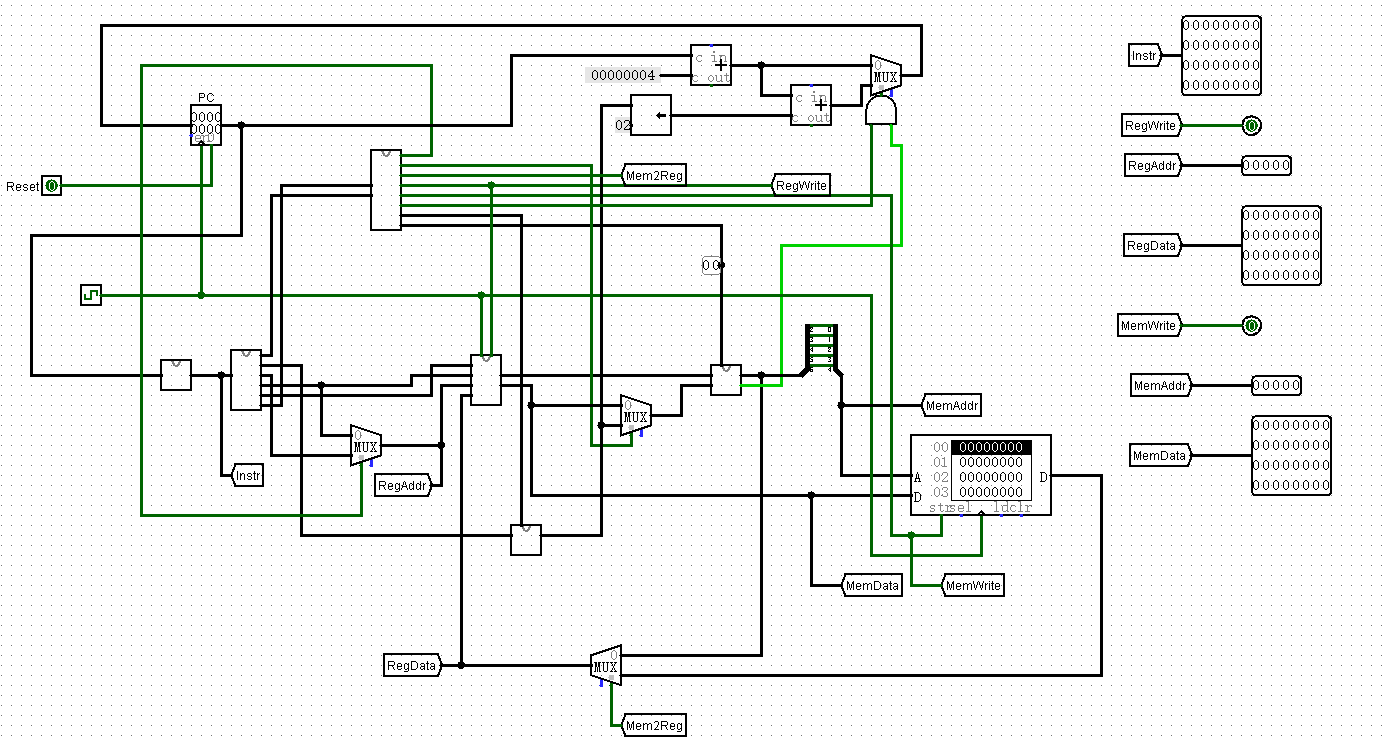


图1 顶层设计图

1. 模块规格

由于Controller较为特殊，在CPU中占有核心地位，将在下一章节描述设计，以下为其他基本元件的设计描述。

ALU功能定义与端口说明：

表1 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算 | 根据选择信号（ALUOp[1:0]）对输入的A, B进行计算, 输出结果C。  00: C = A + B  01: C = A – B  10: C = A | B |

表2 ALU端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | ALUOp[1:0] (I) | ALU运算选择信号 |
| 2 | A[31:0] (I) | 立即数A的输入端口 |
| 3 | B[31:0] (I) | 立即数B的输入端口 |
| 4 | Clk (I) | 时钟信号 |
| 5 | C[31:0] (O) | 运算结果C的输出端口 |
| 6 | Zero (O) | Zero = A==B?1:0 |

GRF功能定义与端口说明：

表3 GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 使能 | 当使能信号有效且在时钟上升沿时将写入数据 |
| 2 | 存储数据 | 将数据 写入指定寄存器 |
| 3 | 读取数据 | 从指定寄存器读取所需数据 |

表4 GRF端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | RA1 (I) | 待读寄存器的地址address1 |
| 2 | RA2 (I) | 待读寄存器的地址address2 |
| 3 | WA (I) | 待写寄存器的地址address3 |
| 4 | WD (I) | 待写入地址为address3寄存器的值 |
| 5 | Clk (I) | 时钟信号 |
| 6 | RD1 (O) | 地址为address1的寄存器的值data1输出端口 |
| 7 | RD2 (O) | 地址为address2的寄存器的值data2输出端口 |

IFU功能定义与端口说明：

表5 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x00000000 |
| 2 | 取指令 | 根据PC从IM中取出指令 |
| 3 | 计算下条指令地址 | 根据不是跳转指令PC = PC + 4，如果是beq，计算跳转后指令地址 |

表6 IFU端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | PC (I) | 输入当前的PC值 |
| 2 | Reset (I) | 输入复位信号 |
| 3 | Clk (I) | 时钟信号 |
| 4 | Instr (O) | 输出取出的指令 |

Splitter功能定义与端口说明：

表7 Splitter功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 分离 | 从Instr中分离出func、immediate、Rd、Rt、Rs、op |

表8 Splitter端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | Instr (I) | 指令的输入端口 |
| 2 | func (O) | R型指令的function输出 |
| 3 | immediate (O) | I型指令的立即数输出 |
| 4 | Rd (O) | 寄存器编号输出 |
| 5 | Rt (O) | 寄存器编号输出 |
| 6 | Rs (O) | 寄存器编号输出 |
| 7 | op (O) | 指令的op输出 |

EXT功能定义与端口说明：

表9 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 立即数扩展 | 根据Extop[1:0]信号对输入立即数A进行扩展  00: 对输入立即数A进行零扩展  01: 对输入立即数A进行符号扩展  10: 对输入立即数A进行加载至高位运算 |

表10 EXT端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | A (I) | 立即数A(16bits)的输入端口 |
| 2 | Extop[1:0] (I) | EXT选择信号 |
| 3 | B (O) | 结果B(32bits)的输出端口 |

DM功能定义与端口说明：

表11 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 使能 | 当使能信号有效且在时钟上升沿时将写入数据 |
| 2 | 写入数据 | 将数据写在指定地址上 |
| 3 | 读取数据 | 将指定地址的数据读取出来 |

表12 DM端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | A (I) | 要写入数据的RAM地址 |
| 2 | D (I) | 写入RAM的数据 |
| 3 | MemWrite (I) | 写操作的使能信号 |
| 4 | Clk (I) | 时钟信号 |
| 5 | RD (O) | RAM读取的数据 |

1. 控制器设计

控制的本质就是一个译码的过程，将指令包含的信息转为CPU各部分的控制信号，由真值表通过与或门阵实现。

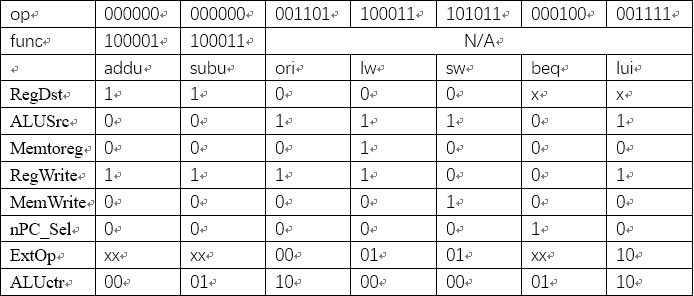
表13 Controller功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 输出控制信息 | 根据输入的6位op和6位func得出各器件控制信息 |

表14 Controller端口说明

|  |  |  |
| --- | --- | --- |
| 序号 | 端口名称 | 功能描述 |
| 1 | Instr (I) | 32位指令的输入 |
| 2 | RegDst (O) | 判断指令类型: 1: R 0: I |
| 3 | ALUSrc (O) | 选择输入ALU的立即数 |
| 4 | Memtoreg (O) | 选择从DM读取写入GRF的数据 |
| 5 | RegWrite (O) | GRF写使能信号 |
| 6 | MemWrite (O) | DM写使能信号 |
| 7 | nPC\_Sel (O) | Beq指令的控制信号 |
| 8 | ExtOp (O) | 扩展立即数选择信号 |
| 9 | ALUctr (O) | ALU运算选择信号 |

表15 Controller真值表



1. CPU测试

测试代码：

lw $t0, 0($0)

lw $t1, 4($0)

loop:

lw $t2, 8($0)

lw $t3, 12($0)

addu $t4, $t0, $t1

subu $t5, $t2, $t3

sw $t4, 8($0)

sw $t5, 12($0)

beq $t5, $0, loop

ori $t6, $t5, 5

sw $t6, 8($0)

测试期望：由DM加载出来$t0=1,$t1=2,$t2=3,$t3=3,$t0+$t1=3写入$t4,$t2-$t3=0写入$t5,将$t4=3写入地址0x00000008,将$t5=0写入地址0x0000000c,之后$t5=$0=0,跳转到loop, 地址0x00000008的数据加载至$t2=3, 地址0x0000000c的数据加载至$t3=0,$t0+$t1=3写入$t4,$t2-$t3=3写入$t5,$t5!=$0不再跳转，$t5|5=7写入$t6,再将$t6=7写入地址0x00000008。

最终，地址0x00000000数据为1，地址0x00000004数据为2，地址0x00000008数据为7，地址0x0000000c数据为3。

1. 思考题

1.若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

答：若PC为30位则执行+1，若为32位则执行+4，本质上没有优劣之别。

2.现在我们的模块中 IM使用ROM， DM使用RAM， GRF使用寄存器，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：合理。IM作为一个指令寄存器，可以直接导入，无需写操作。DM作为一种内存可以用RAM实现，GRF用寄存器实现速度较快。

3.结合上文给出的样例真值表，给出RegDst， ALUSrc， MemtoReg，RegWrite, nPC\_Sel, ExtOp与op和func有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

答：RegDst = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

ALUSrc = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

Memtoreg = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0])

RegWrite = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

| (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

MemWrite = (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

nPC\_Sel = (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ExtOp[0] = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

ExtOp[1] = (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

ALUctr[0] = (~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0]) | (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ALUctr[1] = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

4.充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给出化简后的形式。

答：RegDst = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

ALUSrc = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

Memtoreg = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0])

RegWrite = ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &~func[1] &func[0])) | ((~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0])& (func[5]& ~func[4]& ~func[3] &~func[2] &func[1] &func[0]))

| (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

MemWrite = (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

nPC\_Sel = (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ExtOp[0] = (op[5]& ~op[4]& ~op[3] &~op[2] &op[1] &op[0]) | (op[5]& ~op[4]& op[3] &~op[2] &op[1] &op[0])

ExtOp[1] = (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

ALUctr[0] = (~op[5]& ~op[4]& ~op[3] &~op[2] &~op[1] &~op[0]) | (~op[5]& ~op[4]& ~op[3] &op[2] &~op[1] &~op[0])

ALUctr[1] = (~op[5]& ~op[4]& op[3] &op[2] &~op[1] &op[0]) | (~op[5]& ~op[4]& op[3] &op[2] &op[1] &op[0])

5.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？请给出你的理由。

答：空指令为8’h00000000，即不进行任何操作，控制信号并没有实际操作，所以不用加入真值表。

6.前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

答：对于DM输入地址RA，取出实际地址减去DM起始地址接上RA即可。DM片选信号决定输入地址是RA还是相减的结果。

7.除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。

答：形式验证的优势：

仿真对于超大规模设计太耗费时间，当确认某个功能的仿真是正确的以后，设计实现的每个步骤的结果可以与上个步骤结果做形式比较，不必进行复杂的仿真。

形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了模拟验证的不足。

形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，有可能缩短设计周期。

形式验证的劣势：

不能有效的验证电路的性能，例如电路的延时和功耗，只能和模拟验证相互补充。