**单周期cpu设计文档**

王郁含 16182672

**目录**

1. 设计与测试说明
2. CPU设计文档
3. 模块规格设计
4. 数据通路设计
5. 控制器设计
6. 测试程序
7. 思考题
   1. 设计与测试说明
8. 处理器为32位处理器。
9. 处理器应支持的指令集为：{addu, subu, ori, lw, sw, beq, lui, nop}。
10. nop机器码为0x00000000， 即空指令，不进行任何有效行为（修改寄存器等）
11. addu,subu可以不支持溢出。
12. 处理器为单周期设计。
13. 需要采用模块化和层次化设计。顶层有效的驱动信号要求包括且仅包括：reset (clk 请使用内置时钟模块).
14. 需自行构造测试集，验证设计的正确性。
    1. CPU设计文档
15. 模块规格设计
    * 1. IFU（取指令单元）：
         + - 内部包括 PC（程序计数器）、IM(指令存储器)及相关逻辑。
           - PC 用寄存器实现，应具有复位功能。
           - 起始地址：0x00000000。
           - IM用 ROM 实现，容量为 32bit \* 32。
           - 因 IM 实际地址宽度仅为 5 位，故需要使用恰当的方法将 PC 中储存的地址同 IM 联系起来。

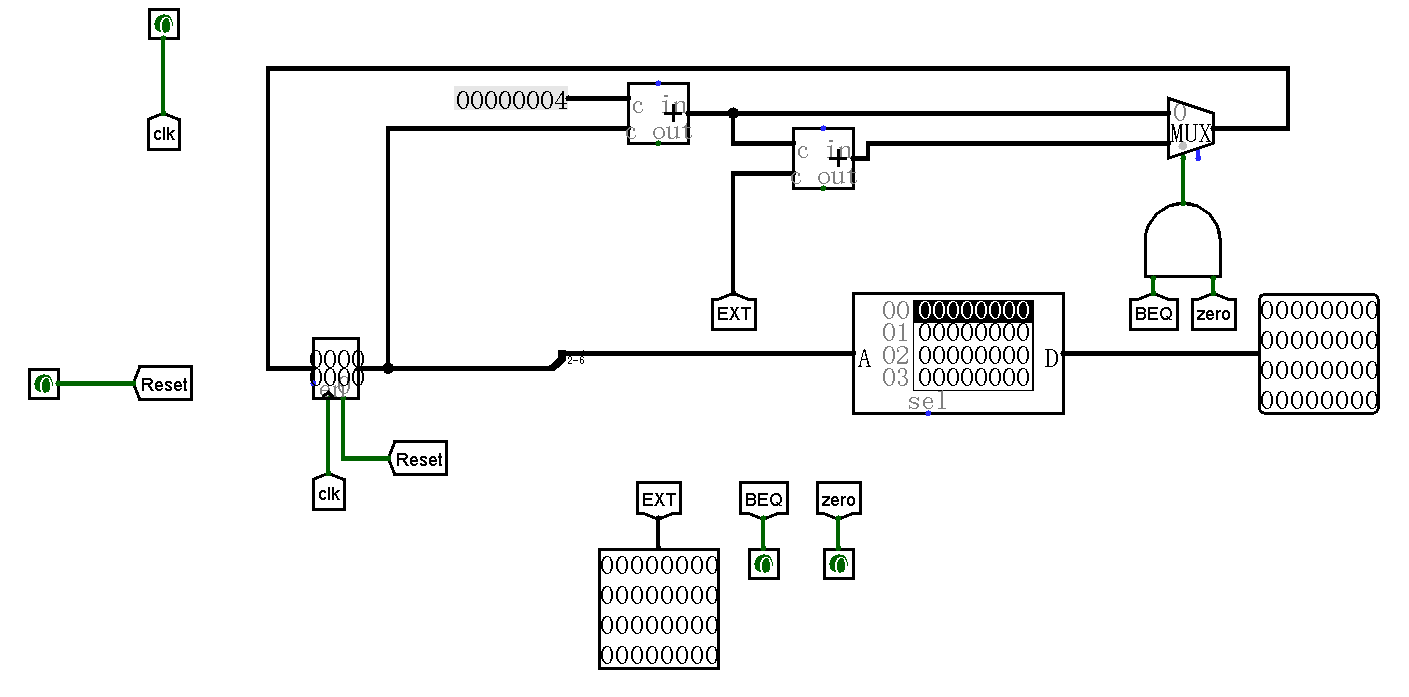


图1 IFU设计

* + 1. GRF（通用寄存器组，也称为寄存器文件、寄存器堆）
       - * 用具有写使能的寄存器实现，寄存器总数为 32 个。
         * 0 号寄存器的值始终保持为 0。其他寄存器初始值均为 0，无需专门设置。

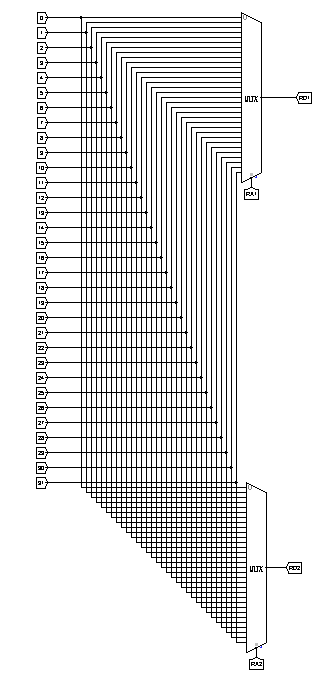


图2 读寄存器设计

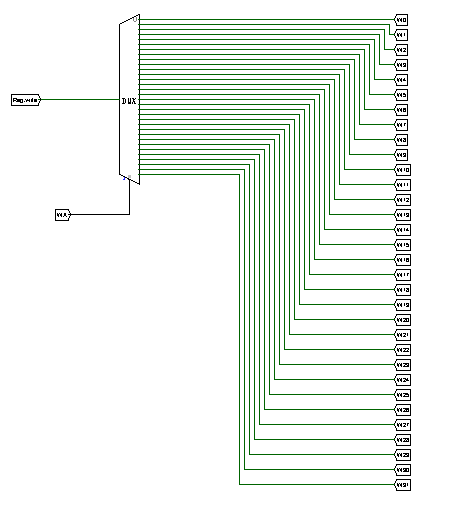


图3 写入寄存器片选信号设计

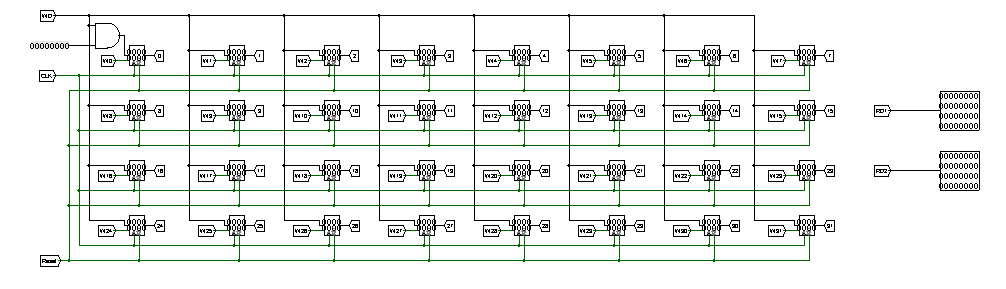


图4 写入寄存器设计（时序逻辑电路）

* + 1. ALU（算术逻辑单元）
       - * 提供 32 位加、减、或运算及大小比较功能。
         * 可以不支持溢出（不检测溢出）。

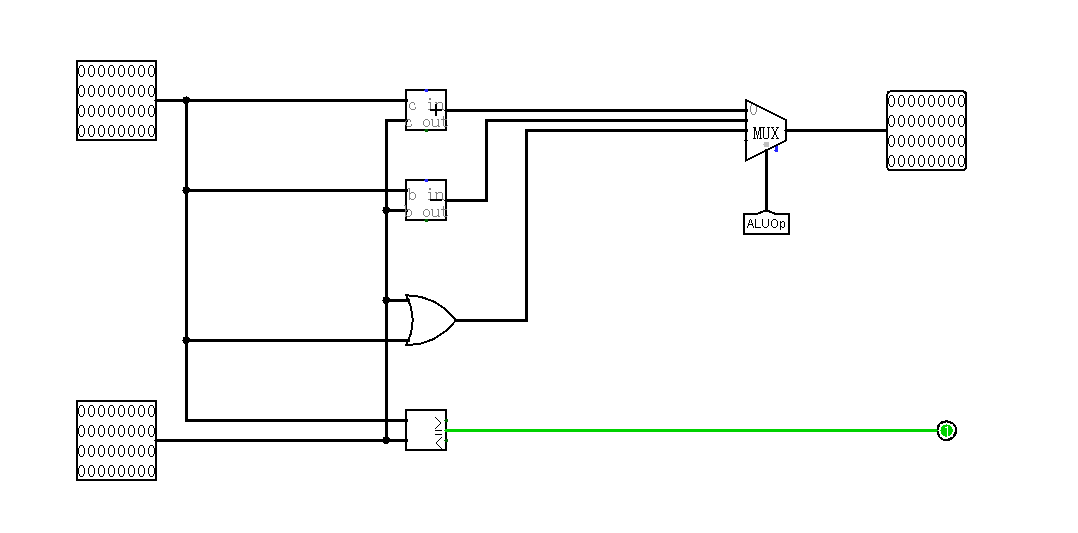


图5 ALU单元设计

* + 1. DM（数据存储器）
       - * 使用 RAM 实现，容量为 32bit \* 32。
         * 起始地址：0x00000000。
         * RAM 应使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports。

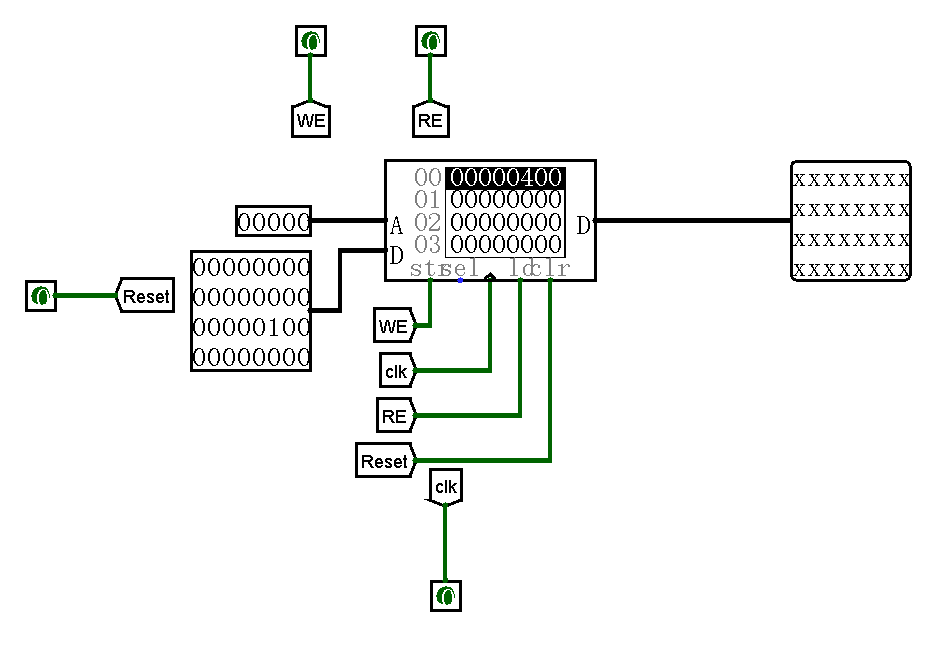


图6 DM设计

* + 1. EXT
       - * 可以使用 logisim 内置的 Bit Extender。

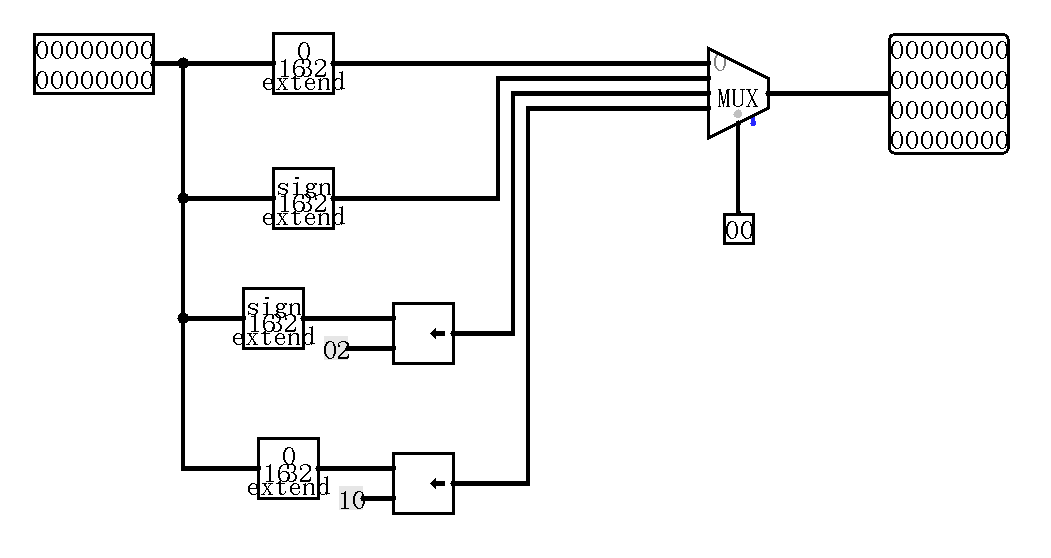


图7 EXT部件设计

1. 数据通路设计

表1 数据通路设计

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | Adder | | PC | IM.A | GRF | | | | ALU | | DM | | EXT | Nadd | |
| A | B | RA1 | RA2 | WA | WD | A | B | Add. | Wdata | A | B |
| R类型 | PC | 4 | Adder | PC | Rs | Rt | Rd | ALU | RF.RD1 | RF.RD2 |  |  |  |  |  |
| ORI | PC | 4 | Adder | PC | Rs |  | Rt | ALU | RF.RD1 | EXT |  |  | Imm16 |  |  |
| LW | PC | 4 | Adder | PC | Rs |  | Rt | DM.RD | RF.RD1 | EXT | ALU |  | Imm16 |  |  |
| SW | PC | 4 | Adder | PC | Rs | Rt |  |  | RF.RD1 | EXT | ALU | RF.RD2 | Imm16 |  |  |
| BEQ | PC | 4 | Adder|Nadd | PC | Rs | Rt |  |  | RF.RD1 | RF.RD2 |  |  | Imm16 | Adder | EXT |
| LUI | PC | 4 | Adder | PC | Rs |  | Rt | ALU | RF.RD1 | EXT |  |  | Imm16 |  |  |

注：Rs:IM.A[25:21]

Rt:IM.A[20:16]

Rd:IM.A[15:11]

1. 控制器设计

表2 控制信号选择

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | n/a | | | | |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| 指令 | addu | subu | ori | lw | sw | beq | lui |
| WAOp | 1 | 1 | 0 | 0 | x | x | 0 |
| WDOp | 0 | 0 | 0 | 1 | x | x | 0 |
| BEQOp | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ALUBOp | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| EXTOp | x | x | 00 | 01 | 01 | 10 | 11 |
| RWE | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| DWE | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| RE | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ALUOP | Add00 | Subtract01 | Or10 | Add00 | Add00 | Cmp11 | Add00 |

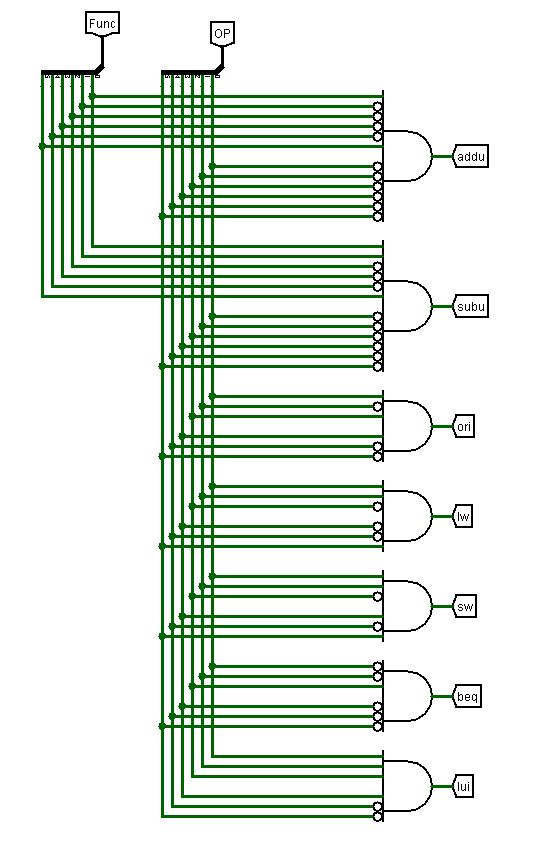


图8 与逻辑设计控制信号

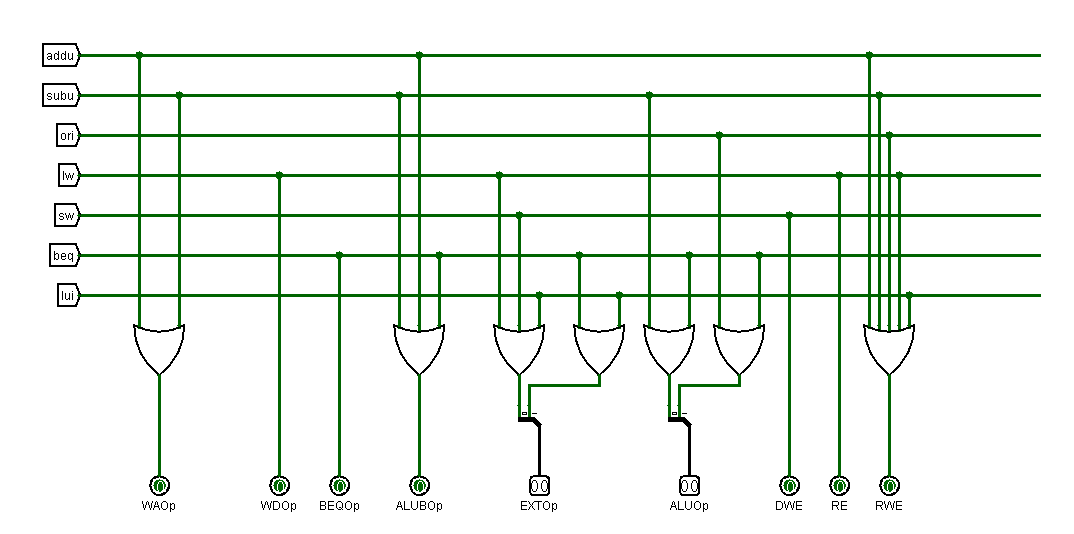


图9 或逻辑设计控制信号

1. 测试程序设计
2. Addu指令

表3 addu指令测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 指令 | 16进制代码 | | | 预期输出 | 说明 | |
| addu $1, $2, $3 | | 00430281 | $1 <= $2 + $3 | | | 正常情况1：无溢出 |
| addu $4, $5, $6 | | 00a60021 | $4 <= $5 + $6 | | | 正常情况2：有溢出 |
| addu $0, $4, $5 | | 00850021 | $0 <= 0 | | | 特殊情况1：存入0号寄存器 |
| addu $6, $0, $7 | | 00073021 | $6 <= $7 | | | 特殊情况2：0号寄存器为第一个操作数 |
| addu $8, $9, $0 | | 01204021 | $8 <= $9 | | | 特殊情况3：0号寄存器为第二个操作数 |

1. Subu指令

表4 subu指令测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 指令 | 16进制代码 | | 预期输出 | | 说明 |
| subu $1, $2, $3 | | 00430823 | $1 <= $2 - $3 | 正常情况1：无溢出 | |
| subu $4, $5, $6 | | 00a62023 | $4 <= $5 - $6 + 0x100000000 | 正常情况2：有溢出 | |
| subu $0, $4, $5 | | 00850023 | $0 <= 0 | 特殊情况1：存入0号寄存器 | |
| subu $6, $0, $7 | | 00073023 | $6 <= 0x100000000 - $7 | 特殊情况2：0号寄存器为被减数 | |
| subu $8, $9, $0 | | 01204023 | $8 <= $9 | 特殊情况3：0号寄存器为减数 | |

1. Ori指令

表5 ori指令测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 指令 | 16进制代码 | | | 预期输出 | | 说明 |
| ori $1, $2, 100 | | 34410064 | $1 <= $2 | 0x00000064 | | 正常情况 | |
| ori $0, $3, 200 | | 346000c8 | $0 <= 0 | | 特殊情况1：存入0号寄存器 | |
| ori $4, $0, 300 | | 3404012c | $4 <= 0x0000012c | | 特殊情况2：0号寄存器为操作数 | |
| ori $5, $6, 0 | | 34c50000 | $5 <= $6 | | 特殊情况2：立即数为0 | |
| ori $7, $0, 0 | | 34070000 | $7 <= 0 | | 特殊情况3：操作数为0和0号寄存器 | |

1. Lw指令

表6 lw指令测试

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 16进制代码 | 预期输出 | 说明 |
| lw $1, 4($2) | 8c410004 | 将DM从4开始向后数$2个数据的数放入到$1中 | 正常情况 |
| lw $0, 6($3) | 8c600006 | $0 <= 0 | 特殊情况：写入0号寄存器 |

1. Sw指令

表7 sw指令测试

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 16进制代码 | 预期输出 | 说明 |
| sw $1, 4($2) | ac410004 | 将$1中的数放入到DM从4开始向后数$2个中 | 正常情况 |

1. Beq指令

表8 beq指令测试

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 16进制代码 | 预期输出 | 说明 |
| beq $0, $0, 1 | 10000001 | 跳转至下下条指令 | 正常情况1：相等时正跳转 |
| addi $1, $0, 1  beq $1, $0, 4 | 20210001  1020fffe | 不跳转，继续执行下一条指令 | 正常情况2：不相等 |
| beq $0, $0, -2 | 1000fffd | 跳转至上上条指令 | 正常情况3：相等时负跳转 |

1. Lui指令

表9 lui指令测试

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 16进制代码 | 预期输出 | 说明 |
| lui $1, 100 | 3c010064 | $1 <= 0x00640000 | 正常情况 |
| lui $0, 200 | 3c0000c8 | $0 <= 0 | 特殊情况1：存入0号寄存器 |
| lui $2, 0 | 3c020000 | $2 <= 0 | 特殊情况2：将0加载至高位 |

1. Nop指令

表10 nop指令测试

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 指令 | 16进制代码 | | | 预期输出 | | 说明 |
| nop | | 0x00000000 | 无 | | 空指令 | |

* 1. 思考题

1. 模块规格设计
   * 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：如果30位pc，同样按字节计算的话，就会使得跳转指令的跳转范围缩小4倍。如果按字计算的话，有可能在出现错误时无法识别，直接执行错误指令。

* + 1. 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为一开始设定好的指令，不会被更改，为了防止误操作被更改，采用ROM。DM即需要被读取也需要被改写，并且数据量较大，所以需要使用RAM。GRF容量较小，需要不断被改写，且寄存器运行速度较快，所以用寄存器。

1. 控制器设计
2. 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：

|  |  |  |
| --- | --- | --- |
| 控制信号 | 逻辑表达式 | 布尔表达式 |
| RegDst | addu | subu |  |
| ALUSrc | ori | lw | sw |  |
| MemtoReg | lw |  |
| RegWrite | addu | subu | ori | lw |  |
| nPC\_Sel | beq |  |
| ExtOp | lw | sw |  |

1. 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：

|  |  |  |
| --- | --- | --- |
| 控制信号 | 逻辑表达式 | 布尔表达式 |
| RegDst | addu | subu | beq |  |
| ALUSrc | ori | lw | sw |  |
| MemtoReg | lw | sw |  |
| RegWrite | addu | subu | ori | lw |  |
| nPC\_Sel | beq |  |
| ExtOp | lw | sw | addu | subu |  |

1. 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：任意一条指令的控制信号均不可能全是0，而nop指令的控制信号不管是与阵还是列阵都是全部是0。即nop不会对应任意一条指令。而当控制信号均为0时的操作为将0写入$0号寄存器，相当于没有操作。

1. 测试程序
2. 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：当text地址从0开始时，data地址从0x00002000开始，此时需要加一个片选信号将地址与0x2000比较，大于的时候就存入DM，存入地址是原地址-0x2000

1. 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：优点：验证者无需考虑如何获得验证的条件；形式验证可以对所有情况进行验证；形式验证有利于尽早发现错误；无需长年累月的经验辅助。

缺点：需要设计者考虑设计步骤，精密计算正确的输入激励步骤；不能完整测试电路的耗能，延时等。