Verilog多周期CPU设计文档

1. CPU设计方案综述
2. 总体设计概述

使用Verilog开发一个简单的单周期CPU，总体概述如下：

1. 此CPU为32位CPU
2. 此CPU为多周期设计
3. 此CPU支持的指令集为：

{addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}

1. nop机器码为0x00000000
2. addu, subu不支持溢出
3. 关键模块定义
4. IM
5. 端口说明

表1-IM端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | PC[31:0] | I | 时钟信号 |
| 2 | instr[31:0] | O | 指令 |

1. 功能定义

表2-IM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 取指令 | 就是取指令 |

1. PC
2. 端口说明

表3-PC端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 复位信号 |
| 3 | Jumpsign | I | 是否进行非PC+4跳转 |
| 4 | JumpAddr[31:0] | I | 非PC+4时NPC值 |
| 5 | freeze | I | 是否暂停 |
| 6 | PC[31:0] | O | 当前指令所在IM地址 |

1. 功能定义

表4-PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 存储指令的地址 | 输出当前指令所在IM地址，并计算下一周期指令所在IM地址。 |

1. GRF
2. 端口说明

表7-GRF端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将32个寄存器中全部清零  1：清零  0：无效 |
| 3 | WE | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| 4 | A1[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1 |
| 5 | A2[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2 |
| 6 | A3[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，作为RD的写入地址 |
| 7 | WD[31:0] | I | 32位写入数据 |
| 8 | RD1[31:0] | O | 输出A1指定的寄存器的32位数据 |
| 9 | RD2[31:0] | O | 输出A2指定的寄存器的32位数据 |

1. 功能定义

表8-GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | reset为1时，将所有寄存器清零 |
| 2 | 读数据 | 将A1和A2地址对应的寄存器的值分别通过RD1和RD2读出 |
| 3 | 写数据 | 当WE为1且时钟上升沿来临时，将WD写入到A3对应的寄存器内部 |

1. ALU
2. 端口说明

表9-ALU端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | A[31:0] | I | 参与运算的第一个数 |
| 2 | B[31:0] | I | 参与运算的第二个数 |
| 3 | AluOp[2:0] | I | 000：无符号加  001：无符号减  010：与  011：或 |
| 4 | res[31:0] | O | A与B做运算后的结果 |

1. 功能定义

表10-ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 加运算 | res = A + B |
| 2 | 减运算 | res = A - B |
| 3 | 与运算 | res = A & B |
| 4 | 或运算 | res = A | B |

1. DM
2. 端口说明

表11-DM端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号  0：无效  1：内存值全部清零 |
| 3 | WE | I | 写使能信号  0：禁止写入  1：允许写入 |
| 4 | MemAddr[31:0] | I | 读取或写入信号地址 |
| 5 | WD[31:0] | I | 32为写入数据 |
| 6 | RD[31:0] | O | 32位读出数据 |

1. 功能定义

表12-DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | 当reset为1时，DM中所有数据清零 |
| 2 | 写入数据 | 当WE有效时，时钟上升沿来临时，WD中数据写入A对应的DM地址中 |
| 3 | 读出数据 | RD永远读出A对应的DM地址中的值 |

1. Controller
2. 端口说明

表15-Controller端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | instr[31:0] | I | instr[31:26],6位控制信号 |
| 2 | PC | I | 指令所在IM地址 |
| 3 | ReadGF1[31:0] | I | 读的寄存器1数据 |
| 4 | ReadGF2[31:0] | I | 读的寄存器2数据 |
| 5 | ReadDM[31:0] | I | 读的数据存储器数据 |
| 6 | ALUresult[31:0] | I | Alu计算结果 |
| 7 | Jumpsign | O | 进行非PC+4跳转 |
| 8 | [31:0] JumpAddr | O | 跳转到的数 |
| 9 | WEgf | O | 寄存器堆写使能 |
| 10 | [4:0] WriteGFadd | O | 写的寄存器地址 |
| 11 | [31:0] WriteGFdata | O | 写的寄存器数据 |
| 12 | [4:0] readGFadd1 | O | 读的寄存器1地址 |
| 13 | [4:0] readGFadd2 | O | 读的寄存器2地址 |
| 14 | WEdm | O | 数据存储器写使能 |
| 15 | [31:0] WriteDMadd | O | 写的数据存储器地址 |
| 16 | [31:0] WriteDMdata | O | 写的数据存储器数据 |
| 17 | [31:0] readDMadd | O | 读的数据存储器地址 |
| 18 | [3:0] ALUopcode | O | Alu运算符 |
| 19 | [31:0] ALUinput1 | O | Alu读入数据1 |
| 20 | [31:0] ALUinput2 | O | Alu读入数据2 |

1. Gap
2. 端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | [31:0] I\_Instr\_out | I | 指令 |
| 2 | [31:0] I\_PC\_out | I | 指令所在地址 |
| 3 | [31:0] D\_Instr\_in | O | 指令 |
| 4 | [31:0] D\_PC\_in | O | 指令所在地址 |
| 5 | [31:0] D\_Instr\_out | I | 指令 |
| 6 | [31:0] D\_PC\_out | I | 指令所在地址 |
| 7 | [31:0] D\_Num1\_out | I | ALU输入数字1 |
| 8 | [31:0] D\_Num2\_out | I | ALU输入数字2 |
| 9 | [31:0] D\_Storedata\_out | I | DM存入数据 |
| 10 | [3:0] D\_ALUopcode\_out | I | ALU运算符 |
| 11 | D\_WEGF\_out | I | 寄存器写使能 |
| 12 | D\_WEDM\_out | I | DM写使能 |
| 13 | [31:0] E\_Instr\_in | O | 指令 |
| 14 | [31:0] E\_PC\_in | O | 指令所在地址 |
| 15 | [31:0] E\_Num1\_in | O | ALU输入数字1 |
| 16 | [31:0] E\_Num2\_in | O | ALU输入数字2 |
| 17 | [31:0] E\_Storedata\_in | O | DM存入数据 |
| 18 | [3:0] E\_ALUopcode\_in | O | ALU运算符 |
| 19 | E\_WEGF\_in | O | 寄存器写使能 |
| 20 | E\_WEDM\_in | O | DM写使能 |
| 21 | [31:0] E\_Instr\_out | I | 指令 |
| 22 | [31:0] E\_PC\_out | I | 指令所在地址 |
| 23 | [31:0] E\_ALUresult\_out | I | 运算结果 |
| 24 | [31:0] E\_Storedata\_out | I | DM存入数据 |
| 25 | E\_WEGF\_out | I | 寄存器写使能 |
| 26 | E\_WEDM\_out | I | DM写使能 |
| 27 | [31:0] M\_Instr\_in | O | 指令 |
| 28 | [31:0] M\_PC\_in | O | 指令所在地址 |
| 29 | [31:0] M\_ALUresult\_in | O | 运算结果 |
| 30 | [31:0] M\_Storedata\_in | O | DM存入数据 |
| 31 | M\_WEGF\_in | O | 寄存器写使能 |
| 32 | M\_WEDM\_in | O | DM写使能 |
| 33 | [31:0] M\_Instr\_out | I | 指令 |
| 34 | [31:0] M\_PC\_out | I | 指令所在地址 |
| 35 | [31:0] M\_ALUresult\_out | I | 运算结果 |
| 36 | [31:0] M\_DMdata\_out | I | DM存入数据 |
| 37 | M\_WEGF\_out | I | 寄存器写使能 |
| 38 | [31:0] W\_Instr\_in | O | 指令 |
| 39 | [31:0] W\_PC\_in | O | 指令所在地址 |
| 40 | [31:0] W\_ALUresult\_in | O | 运算结果 |
| 41 | [31:0] W\_DMdata\_in | O | DM存入数据 |
| 42 | W\_WEGF\_in | O | 寄存器写使能 |

1. 测试方案
2. 测试代码：

ori $1,11

ori $2,22

ori $3,33

lui $4,12

lui $5,23

lui $6,24

lui $7,25

lui $8,34

lui $9,12

addu $10,$9,$9

addu $11,$2,$3

addu $12,$5,$6

subu $13,$3,$5

subu $14,$5,$4

subu $15,$2,$6

nop

lui $16,12

beq $9,$16, next #应跳转

nop

lui $1,1

lui $2,1

lui $3,1

lui $4,1

haha:

lui $5,1

lui $6,1

lui $7,1

lui $8,1

lui $9,1

lui $10,1

next:

beq $1,$2,haha #应不跳转，否则死循环

sw $1,0($0)

sw $2,4($0)

sw $3,8($0)

sw $4,12($0)

sw $5,16($0)

sw $6,20($0)

sw $7,24($0)

lw $17,0($0)

lw $18,4($0)

jal ok

lw $19, 8($0)

jal end

ok:

lw $0,0($0)

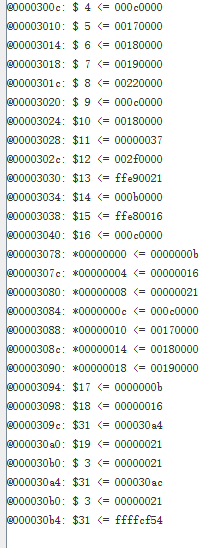
jr $31

end:

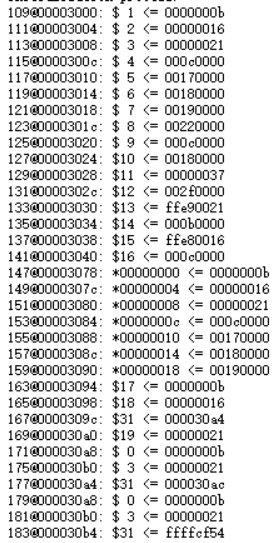
subu $3,$3,$0

subu $31,$0, $31

1. MARS中运行结果

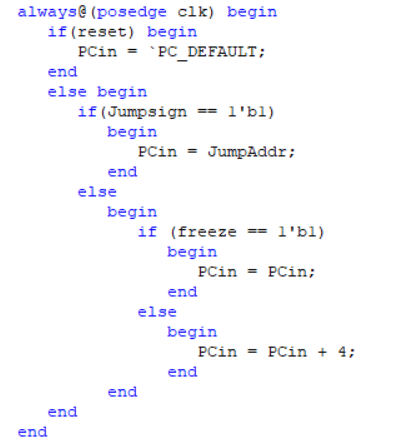


1. 该CPU运行输出结果



1. 思考题
2. **流水线冒险**

**在采用本节所述的控制冒险处理方式下，PC的值应当如何被更新？请从数据通路和控制信号两方面进行说明。**



数据通路：

PC：特殊跳转J信号，特殊跳转J地址，暂停freeze信号。

控制信号：

均来自D级控制

**对于jal等需要将指令地址写入寄存器的指令，为什么需要回写PC+8？**

含有延迟槽，测试的时候默认后面有nop，故为PC+8.

特殊处理（其实我不认同PC+8）

1. **数据冒险的分析**

**为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由ALU或者DM等部件来提供数据？**

因为ALU，DM等均为组合逻辑部件，与时钟无关，不应当有数据复用，否则会落后/提前一个时钟周期。按照我的结构：

上级寄存器读出 -> 组合逻辑部件读入-> 组合逻辑部件输出->下级寄存器输入

1. **AT法处理流水线数据冒险**

**“转发（旁路）机制的构造”中的Thinking 1-4**

1. NOP的时候，会影响同一指令多次运行。（其实我觉得完全不应该这么思考。。。。。）
2. 这样写gf就可以省一个周期的nop
3. 0写入均为0，如写0号，则需要在之前所有转发特判，如果需要读取0号，则强制为0
4. 就是转发的优先级。以jr为例，在D级就要产生写入的NPC数据，此时若E级和M级要求写入同一个寄存器，导致数据冲突，则需要选择E级流水线的数据

**在AT方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为0”，但在CPU写入GRF的时候，是有一个we信号来控制是否要写入的。为何在AT方法中不需要特判we呢？为了用且仅用A和T完成转发，在翻译出A的时候，要结合we做什么操作呢？**

如果不需要写寄存器，直接将A译码为为0，这样甚至可以直接省略we。

1. **在线测试相关说明**

**在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？**

L类，读-写类，S类，beq类，j类，排列组合即可。

**如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。**

1. 见上题，均为手造。