Verilog多周期CPU设计文档

作者：李健健

1. CPU设计方案综述
2. 总体设计概述

使用Verilog开发一个简单的多周期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. PC
5. 端口说明

表1-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 | NPC | I | 下一条指令所在IM地址 |
| 4 | PC | O | 当前指令所在IM地址 |

1. 功能定义

表2-PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 存储指令的地址 | 保存当前执行指令在IM中的地址 |

1. IM
2. 端口说明

表3-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4-IM功能定义

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

1. IFID
2. 端口说明

表5-IDIF功能定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | en | I | 使能信号 |
| 3 | reset | I | 同步复位信号 |
| 4 | PCF[31:0] | I | PC在F级的值 |
| 5 | InstrF[31:0] | I | instr在F级的值 |
| 6 | PCD[31:0] | O | PC在D级的值 |
| 7 | InstrD[31:0] | O | instr在D级的值 |

1. 功能定义

表6-IFID功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 存储流水线值 | 存储流水线值 |

1. NPC
2. 端口说明

表7-NPC端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | branch | I | 分支信号 |
| 2 | JType | I | 跳转信号 |
| 3 | JReg | I | 判断指令是否需要跳转寄存器 |
| 4 | PCF[31:0] | I | F级PC值 |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | PCD[31:0] | I | D级PC值 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | RegJump[31:0] | I | 跳转寄存器中地址值 |
| 7 | imm26D[25:0] | I | D级的26位立即数 |
| 8 | NPC[31:0] | O | 根据各种指令计算出的下一个PC值 |

1. 功能定义

表8-NPC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 计算下一个PC的值 |  |

1. GRF
2. 端口说明

表9-GRF端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 同步复位信号，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10-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. CMP
2. 端口说明

表11-CMP端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | A[31:0] | I | 操作数A |
| 2 | B[31:0] | I | 操作数B |
| 3 | eq | O | A==B？ |
| 4 | eqz | O | A==0？ |
| 5 | ltz | O | A<0? |

1. 功能描述

表12-CMP功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 判断A和B是否相等 | 若A等于B，eq置一，否则置零 |
| 2 | 判断A是否等于0 | 若A等于0，eqz置一，否则置零 |
| 3 | 判断A是否小于0 | 若A小于0，ltz置一，否则置零 |

1. EXT
2. 端口说明

表13-EXT端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | imm16[15:0] | I | 代扩展的16位信号 |
| 2 | sign | I | 无符号或符号扩展选择信号  0：无符号扩展  1：符号扩展 |
| 3 | imm32[31:0] | O | 扩展后的32位的信号 |

1. 功能定义

表14-EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 无符号扩展 | 当sign为0时，将imm16无符号扩展输出 |
| 2 | 符号扩展 | 当sign为1时，将imm16符号扩展输出 |

1. IDEX
2. 功能定义

表15-IDEX功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保留ID/EX级流水线信息 |

1. ALU
2. 端口说明

表16-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 | | 决定ALU做何种操作  0000：无符号加  0001：无符号减  0010：与  0011：或  0100：将B[15:0]做为res[31:16],res[15:0]=0 |
| 5 | res[31:0] | | O | | A与B做运算后的结果 |

1. 功能定义

表17-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 |
| 5 | 加载高位运算 | res = {B[15:0], 16’h0} |

1. EXMEM
2. 功能定义

表18-EXMEM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保留EX/MEM级流水线信息 |

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 | witdth | I | 读写位宽 | |
| 5 | LoadSign | I | 读写时是否带符号 | |
| 6 | RD[31:0] | O | 32位读出数据 |
| 4 | addr[31:0] | I | 读取或写入信号地址 |
| 5 |  |  |  | |
|  |  |  |  | |

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 | eq | I | RegRead1和RegRead是否相等 |
| 3 | WeGrf | O | GRF写使能信号  0：禁止写入  1：允许写入 |
| 4 | WeDm | O | DM的写入信号  0：禁止写入  1：允许写入 |
| 5 | RegDst[1:0] | O | GRF写入地址选择信号  0：Rd  1：Rt |
| 6 | WhichtoReg[1:0] | O | 将何种数据写入GRF？  00：ALU计算结果  01：DM读出信号  11：upperImm |
| 7 | AluSrc | O | 参与ALU运算的第二个数，来自GRF还是imm  0：来自GRF  1：imm |
| 8 | AluOp[2:0] | O | ALU的控制信号 |
| 9 | sign | O | 是否对imm16进行符号扩展  0：不进行符号扩展  1：进行符号扩展 |
| 10 | branch | O | instr是否为beq信号  0：不是  1：是 |
| 11 | JType | O | 是不是J型指令，只有j和jal是 |
| 12 | jr | O | 是不是jr指令 |

1. 真值表

表16-Controller内部真值对应

he

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 端口 | addu | subu | ori | lw | sw | lui | beq | jal | jr |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 001111 | 000100 | 000011 | 000000 |
| func | 100001 | 100011 |  |  |  |  |  |  | 001000 |
| WeGrf | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| WeDm | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| RegDst | 00 | 00 | 01 | 01 | 00 | 01 | 00 | 10 | 00 |
| WhichtoReg | 00 | 00 | 00 | 01 | 00 | 00 | 00 | 10 | 00 |
| AluSrc | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| AluOp | 000 | 001 | 011 | 000 | 000 | 100 | 000 | 000 | 000 |
| sign | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| branch | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| JType | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| jr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1. 测试方案
2. 测试代码：

.text

addi $t1, $0, 4

addi $t0, $0, 4

sw $t0, 0($0)

addi $t0, $0, 8

sw $t0, 4($0)

addi $t0, $0, 12

sw $t0, 8($0)

lw $t0, 4($0)

addu $t0, $t0, $t1

lw $t0, 4($t0)

ori $t0, 8

lw $t0, 12($v0)

sw $t0, 4($0)

lw $t0, 12($v0)

sw $t0, 0($t0)

addi $t0, $v0, 4

beq $t0, $t0, Tag1

addi $s1, $v0, 256

Tag1:

lui $t0, 100

beq $t0, $t0, Tag2

addi $s1, $v0, 256

Tag2:

ori $t0, 100

beq $t0, $t0, Tag3

addi $s1, $v0, 256

Tag3:

lw $t0, 4($v0)

beq $t0, $t0, Tag4

addi $s1, $v0, 256

Tag4:

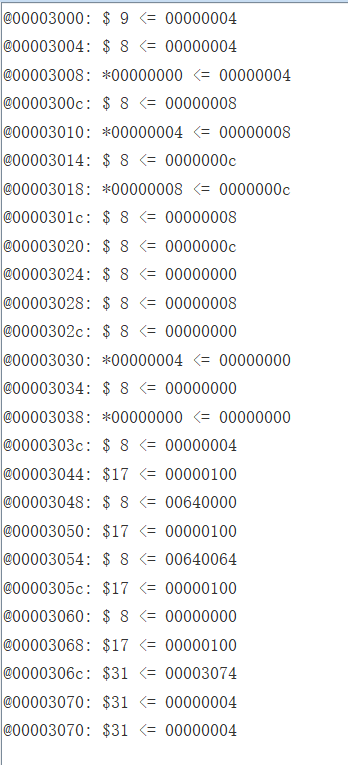
jal Tag5

Tag5:

addi $31, $0, 4

jr $31

1. MARS中运行结果



1. 该CPU运行输出结果

55@00003000: $ 9 <= 00000004

65@00003004: $ 8 <= 00000004

65@00003008: \*00000000 <= 00000004

85@0000300c: $ 8 <= 00000008

85@00003010: \*00000004 <= 00000008

105@00003014: $ 8 <= 0000000c

105@00003018: \*00000008 <= 0000000c

125@0000301c: $ 8 <= 00000008

145@00003020: $ 8 <= 0000000c

155@00003024: $ 8 <= 00000000

175@00003028: $ 8 <= 00000008

185@0000302c: $ 8 <= 00000000

185@00003030: \*00000004 <= 00000000

205@00003034: $ 8 <= 00000000

215@00003038: \*00000000 <= 00000000

235@0000303c: $ 8 <= 00000004

265@00003044: $17 <= 00000100

275@00003048: $ 8 <= 00640000

305@00003050: $17 <= 00000100

315@00003054: $ 8 <= 00640064

345@0000305c: $17 <= 00000100

355@00003060: $ 8 <= 00000000

395@00003068: $17 <= 00000100

405@0000306c: $31 <= 00003074

415@00003070: $31 <= 00000004

425@00003070: $31 <= 00000004

465@00000004: $ 8 <= 00000004

465@00000008: \*00000000 <= 00000004

485@0000000c: $ 8 <= 00000008

485@00000010: \*00000004 <= 00000008

505@00000014: $ 8 <= 0000000c

505@00000018: \*00000008 <= 0000000c

525@0000001c: $ 8 <= 00000008

545@00000020: $ 8 <= 0000000c

555@00000024: $ 8 <= 00000000

575@00000028: $ 8 <= 00000008

585@0000002c: $ 8 <= 00000000

585@00000030: \*00000004 <= 00000000

605@00000034: $ 8 <= 00000000

615@00000038: \*00000000 <= 00000000

635@0000003c: $ 8 <= 00000004

665@00000044: $17 <= 00000100

675@00000048: $ 8 <= 00640000

705@00000050: $17 <= 00000100

715@00000054: $ 8 <= 00640064

745@0000005c: $17 <= 00000100

755@00000060: $ 8 <= 00000000

795@00000068: $17 <= 00000100

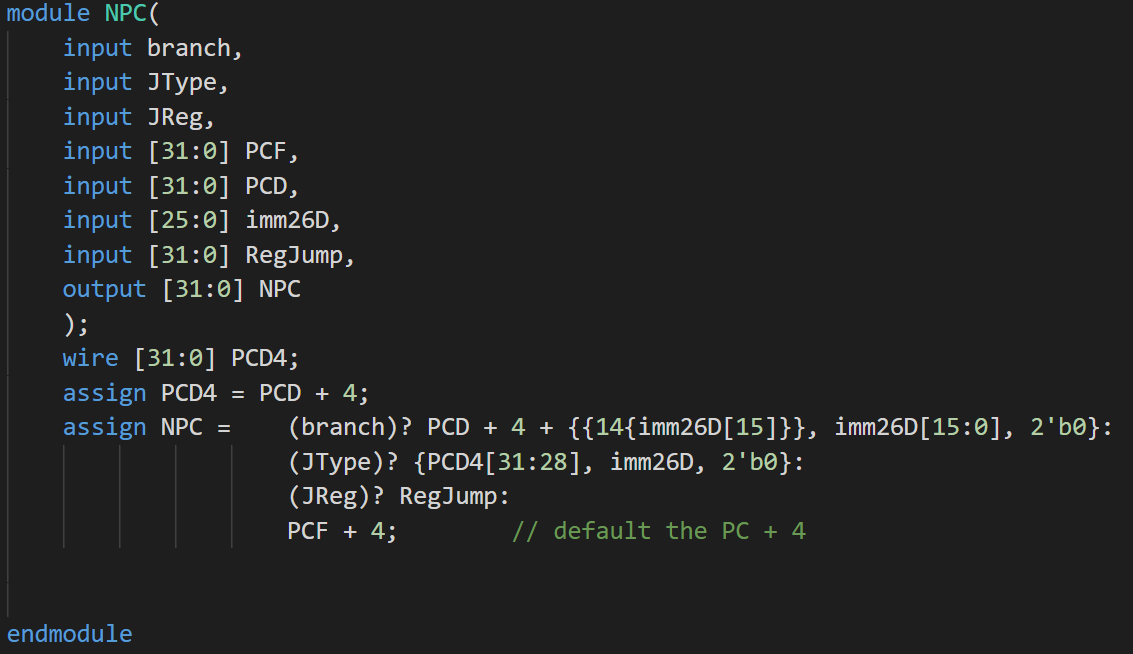
805@0000006c: $31 <= 00000074

815@00000070: $31 <= 00000004

825@00003070: $31 <= 00000004

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：PC+4、branch型指令、JumpRegister指令和J型指令。

控制信号：

controller解码出应当选择的NPC值即可。

**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，后面有nop。

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？**

转发的目的：若算出来，才转发，否则暂停。这是使用“暂停-转发”策略下的核心思想。

直接转发的话，“Brutal-Forward”，也不是不行，但是一来课程组禁止这样，二来转发控制会变得复杂。

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**

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？

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。