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：{

LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、  
SUB、 SUBU、 MULT、 MULTU、 DIV、 DIVU、 SLL、 SRL、 SRA、 SLLV、SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO}

1. nop机器码为0x00000000
2. addu, subu, addiu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 | addr[31:0] | I | 读取或写入信号地址 |
| 7 | WD[31:0] | I | 写入Mem的值 |
| 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 | eq | I | RegRead1和RegRead2是否相等 |
| 3 | sign | O | Ext是否进行符号扩展 |
| 4 | branch | O | 是否进行分支 |
| 5 | JType | O | J型信号 |
| 6 | JReg | O | 是否读取寄存器值作为NPC |
| 7 | WDSelD | O | 若D级生成写入寄存器的值的选择信号。 |
| 8 | A3DE | O | 写入寄存器的地址 |
| 9 | ALUOp | O | ALU怎么算 |
| 10 | ALUASel | O | ALU第一个操作数的选择信号 |
| 11 | ALUBSel | O | ALU第二个操作数的选择信号 |
| 12 | WDSelE | O | E级写入寄存器的值的选择信号 |
| 13 | WEMem | O | DM写使能信号  0：禁止写入  1：允许写入 |
| 14 | width | O | 写入Mem的位宽 |
| 15 | LoadSign | O | 是否load带符号的值 |
| 16 | WDSelM | O | M级生成写入寄存器的值的选择信号 |
| 17 | D1Use | O | D级是否读取rs |
| 18 | D2Use | O | D级是否读取rt |
| 19 | E1Use | O | E级是否读取rs |
| 20 | D2Use | O | E级是否读取rt |
| 21 | M2Use | O | M级是否读取rt |

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 |
| WEMem | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ALUBSel | 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 |
| JReg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1. 转发暂停控制

不同于教程所采用的方法，我使用的是一种方法。面对不同的情况，分别将指令暂停在D级或E级。

转发和暂停之间有优先级，

暂停在D级和E级有优先级。

每一级都将要写入到寄存器的数据流水，当一级要使用的数据冲突时，发现要写入的数据为高阻态，则暂停。

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

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 重置信号 |
| 3 | A1D[4:0] | I | D级读取rs的序号 |
| 4 | A2D[4:0] | I | D级读取rt的序号 |
| 5 | RD1D[31:0] | I | D级读取rs的值 |
| 6 | RD2D[31:0] | I | D级读取rt的值 |
| 7 | D1Use | I | D级是否在使用rs |
| 8 | D2Use | I | D级是否在使用rt |
| 9 | A1E[4:0] | I | E级读取rs的序号 |
| 10 | A2E[4:0] | I | E级读取rt的序号 |
| 11 | RD1E[31:0] | I | E级读取rs的值 |
| 12 | RD2E[31:0] | I | E级读取rt的值 |
| 13 | E1Use | I | E级是否使用rs |
| 14 | E2Use | I | E级是否使用rt |
| 15 | A3E[4:0] | I | 写入寄存器的序号 |
| 16 | WDE[31:0] | I | E级写入寄存器的值 |
| 17 | A2M[4:0] | I | M级读取rt的序号 |
| 18 | RD2M[31:0] | I | M级读取到寄存器的值 |
| 19 | M2Use | I | M级是否在使用rt |
| 20 | A3M[4:0] | I | M级写入寄存器的信号 |
| 21 | WDM[31:0] | I | M级写入寄存器的值 |
| 22 | A3W[4:0] | I | W级写入寄存器的序号 |
| 23 | WDW[31:0] | I | W级写入寄存器的值 |
| 24 | FwdD1[31:0] | O | 转发给D级的值 |
| 25 | FwdD2[31:0] | O | 转发给D级的值 |
| 26 | FwdE1[31:0] | O | 转发给E级的值 |
| 27 | FwdE2[31:0] | O | 转发给E级的值 |
| 28 | FwdM2[31:0] | O | 转发给M级的值 |
| 29 | EnPC | O | PC的使能信号 |
| 30 | EnIFID | O | IFID的使能信号 |
| 31 | EnIDEX | O | IDEX的使能信号 |
| 32 | FlushIDEX | O | IDEX的清零信号 |
| 33 | FlushEXMEM | O | EXMEM的清零信号 |

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:

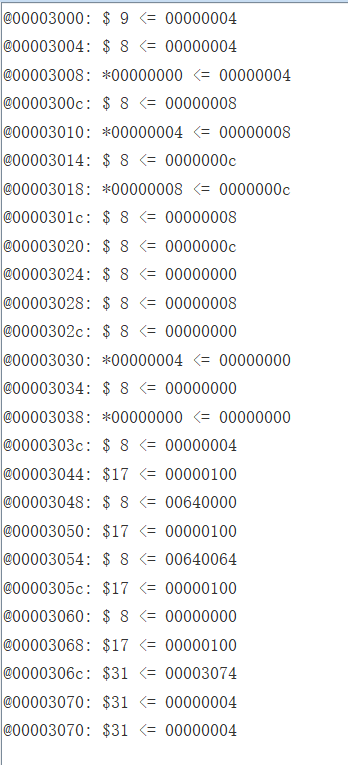
jal Tag5

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

1. 思考题
2. **为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI、LO寄存器？**

因为要遵循“高内聚，低耦合”的设计思想。MDU有和ALU不同的特性，是个时钟原件。

有独立的HI和LO，操作会更加灵活，多加2个寄存器，得到更大的优势。

1. **参照你对延迟槽的理解，试解释“乘除槽”。**

利用编译调度，把本应暂停的周期利用起来。

1. **举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。**

字符串读写、更改。

因为一个字符的ASCII码就一个字节，不需要读一个字。

1. **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？**

主要还是使用MDU的指令之间的冲突。

在D级检测是否该指令要使用MDU，

暂停的条件是要使用MDU并且MDU处于start或busy的状态。

测试样例：

乘除运算后面跟上各种要使用MDU的指令。

1. **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。**

**第一步：指令分类**

**{**

**load:** LB、LBU、LH、LHU、LW

**store:** SB、SH、SW

**cal\_r:**

ADD、ADDU、SUB、SUBU、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、SLT、SLTU

**cal\_i:**

SLL、SRL、SRA、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLTI、SLTIU

**branch\_rs&rt:** BEQ、BNE

**branch\_rs:** BLEZ、BGTZ、BLTZ、BGEZ

**JType:** J、JAL

**JReg:** JALR、JR

**MDU: MULT、 MULTU、 DIV、 DIVU、**

**mf: mfhi, mflo**

**mt: mthi, mtlo**

**}**

**第二步，根据分类指令确定冲突指令：**



**第三步：手动构造冲突指令**

1. **为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？**
   * + 1. 指令分类

同一类别的指令，功能相似，32位的instr也相似。

处理数据冲突时，很像。

* + - 1. 画CPU

帮助更加直观地理解。