Verilog 多周期 CPU 设计文档

一、 CPU 设计方案综述

（一）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}
4. nop 机器码为 0x00000000
5. addu, subu 不支持溢出

（二）关键模块定义

1. IM

（1） 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

（2） 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

2. PC

（1） 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

（2） 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。

3. GRF

（1） 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

（2） 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

4. ALU

(1)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

(2)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

5. DM

(1)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

(2)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

6. Controller

(1)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

7. Gap

(1)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

二、 测试方案

（1）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
```

（2）MARS 中运行结果

```
@0000300c: $ 4 <= 000c0000
@00003010: $ 5 <= 00170000
@00003014: $ 6 <= 00180000
@00003018: $ 7 <= 00190000
@0000301c: $ 8 <= 00220000
@00003020: $ 9 <= 000c0000
@00003024: $10 <= 00180000
@00003028: $11 <= 00000037
@0000302c: $12 <= 002f0000
@00003030: $13 <= ffe90021
@00003034: $14 <= 000b0000
@00003038: $15 <= ffe80016
@00003040: $16 <= 000c0000
@00003078: *00000000 <= 0000000b
@0000307c: *00000004 <= 00000016
@00003080: *00000008 <= 00000021
@00003084: *0000000c <= 000c0000
@00003088: *00000010 <= 00170000
@0000308c: *00000014 <= 00180000
@00003090: *00000018 <= 00190000
@00003094: $17 <= 0000000b
@00003098: $18 <= 00000016
@0000309c: $31 <= 000030a4
@000030a0: $19 <= 00000021
@000030b0: $ 3 <= 00000021
@000030a4: $31 <= 000030ac
@000030b0: $ 3 <= 00000021
@000030b4: $31 <= ffffcf54
```

（3）该 CPU 运行输出结果

```
109@00003000: $ 1 <= 0000000b
111@00003004: $ 2 <= 00000016
113@00003008: $ 3 <= 00000021
115@0000300c: $ 4 <= 000c0000
117@00003010: $ 5 <= 00170000
119@00003014: $ 6 <= 00180000
121@00003018: $ 7 <= 00190000
123@0000301c: $ 8 <= 00220000
125@00003020: $ 9 <= 000c0000
127@00003024: $10 <= 00180000
129@00003028: $11 <= 00000037
131@0000302c: $12 <= 002f0000
133@00003030: $13 <= ffe90021
135@00003034: $14 <= 000b0000
137@00003038: $15 <= ffe80016
141@00003040: $16 <= 000c0000
147@00003078: *00000000 <= 0000000b
149@0000307c: *00000004 <= 00000016
151@00003080: *00000008 <= 00000021
153@00003084: *0000000c <= 000c0000
155@00003088: *00000010 <= 00170000
157@0000308c: *00000014 <= 00180000
159@00003090: *00000018 <= 00190000
163@00003094: $17 <= 0000000b
165@00003098: $18 <= 00000016
167@0000309c: $31 <= 000030a4
169@000030a0: $19 <= 00000021
171@000030a8: $ 0 <= 0000000b
175@000030b0: $ 3 <= 00000021
177@000030a4: $31 <= 000030ac
179@000030a8: $ 0 <= 0000000b
181@000030b0: $ 3 <= 00000021
183@000030b4: $31 <= ffffcf54
```

三、 思考题

（一） 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。

```
always@(posedge clk) begin
    if(reset) begin
        PCin = `PC_DEFAULT;
    end
    else begin
        if(Jumpsign == 1'b1)
            begin
                PCin = JumpAddr;
            end
        else
            begin
                if (freeze == 1'b1)
                    begin
                        PCin = PCin;
                    end
                else
                    begin
                        PCin = PCin + 4;
                    end
            end
    end
end
```

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）

（二） 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

（三） 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。

（四） 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。