CPU 设计文档

一、模块规格

1.IFU

模块接口

信号名	方向	描述
IfBeq	I	当前指令是否为 beq 指令的标志 1：当前指令是 beq 指令 0：当前指令不是 beq 指令
Zero	I	ALU 计算结果为 0 的标志 1：计算结果是 0 0：计算结果不是 0
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
Instr[31:0]	O	32 位 MIPS 指令

功能定义

序号	功能	描述
1	复位	当复位信号有效时，PC 被设置为 0x00000000
2	取指令	根据 PC 从 IM 中取出指令
3	计算下一条指令的地址	如果当前的指令不是 beq 指令，PC=PC+4 如果当前指令是 beq 指令，且 Zero=1 则 PC=PC+4+offset

2.GPR

模块接口

信号名	方向	描述
RegData[31:0]	I	32 位写入数据
RegWrite	I	写入控制信号
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
A1[4:0]	I	读寄存器地址 1
A2[4:0]	I	读寄存器地址 2
RegAddr[4:0]	I	写寄存器地址
Rd1[31:0]	O	32 位输出数据 1
Rd2[31:0]	O	32 位输出数据 2

功能定义

序号	功能	描述
1	复位	当复位信号有效时，所有寄存器被设置为 0x00000000
2	读寄存器	根据输入的寄存器地址读出数据
3	写寄存器	根据输入的地址，把输入的数据写进选中的寄存器

3.ALU

模块接口

信号名	方向	描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
F[2:0]	I	控制信号 000：加 001：减 010：或运算 11：比较大小
C[31:0]	O	32 位数据输出

功能定义

序号	功能	描述
1	加	C=A+B
2	减	C=A-B
3	或	C=A|B
4	比较大小	If(A<B) C=1; else C=0;

4.EXT

模块接口

信号名	方向	描述
A[15:0]	I	16 位数据输入
Extop[1:0]	I	控制信号 00：无符号扩展 01：有符号扩展 10：低位补 0
B[31:0]	O	32 位数据输出

功能定义

序号	功能	描述
1	无符号扩展	无符号扩展
2	有符号扩展	有符号扩展

3	低位补 0	低 16 位补 0

5.DM

模块接口

信号名	方向	描述
MemData[31:0]	I	32 位写入数据
MemWrite	I	写入控制信号 1：写操作
Memtoreg	I	读出控制信号 1：读操作
MemAddr	I	5 位写入地址
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
Out[31:0]	O	32 位数据输出

功能定义

序号	功能	描述
1	复位	当复位信号有效时，所有数据被设置为 0x00000000
2	读	根据输入的寄存器地址读出数据
3	写	根据输入的地址，把输入的数据写入

6.Controller

模块接口

信号名	方向	描述
opcode[5:0]	I	六位输入
func[5:0]	I	六位输入
Regdst	O	写地址控制 1：写寄存器的目标寄存器来自 rd 字段（位于 15:11） 0：写寄存器的目标寄存器来自 rt 字段（位于 20:16）
Alusrc	O	ALU 第二操作数选择控制 1：第二个 ALU 操作数为指令低 16 位的符号拓展 0：第二个 ALU 操作数来自寄存器堆的第二个输出（读数据 2）

Memtoreg	0	DM 读控制 1：写入寄存器的数据来自数据存储器 0：写入寄存器的数据来自 ALU
RegWrite	0	GRF 写入控制信号 1：寄存器堆写使能有效 0：无
MemWrite	0	DM 写入控制信号 1：数据存储器写使能有效 0：无
nPC_sel	0	Beq 指令标志 1：指令为 beq 0：指令不是 beq
Extop	0	控制 ext 拓展方式 00：无符号扩展 01：高 16 位补 0 10：低 16 位补 0
Aluop[2:0]	0	控制 cpu 进行相应的运算 000：加 001：减 010：或 011：大小比较

二、控制器设计

Func[5:0]	100001	100011						000000
Opcode[5:0]	000000	000000	001101	100011	101011	000100	001111	000000
	addu	subu	ori	lw	sw	beq	lui	nop
RegDst	1	1	0	0	X	X	0	X
RegWrite	1	1	1	1	0	0	1	X
ALUsrc	0	0	1	1	1	0	1	X
MemtoReg	0	0	0	1	X	X	X	X
MemWrite	0	0	0	0	1	0	0	X
nPC_sel	0	0	0	0	0	1	0	X

Extop[1:0]	XX	XX	00	01	01	XX	10	XX
Aluop[1:0]	000	001	010	XXX	XXX	XXX	XXX	XXX

三、测试 CPU

测试程序：

```
lui $t1,0xf
beq $zero,$t1,labe1
ori $t1,$t1,0xa
bgez $t1,labe1
labe1:
addu $t4,$t1,$t1
bne $t1,$t1,labe2
subu $t2,$t4,$t1
nop
nop
nop
sw $t2,8($zero)
lw $t3,8($zero)
sw $t4,4($zero)
andi $t5,$t2,0xf
sw $t5,16($zero)
labe2:
```

测试期望：

$9 存入 0x000f_000a　　$10 存入 0x000f_000a　　$11 存入 0x000f_000a

$12 存入 0x001e_0014　　$13 存入 0x0000_000a

地址 0x0000_0004 存入 0x001e_0014　　地址 0x0000_0008 存入 0x000f_000a

地址 0x0001_0000 存入 0x0000_000a

思考题

1.在上个学年的计组课程中，PC（程序计数器）位数被规定为30位，试分析其与32位PC的优劣。

解：32位PC不需要扩展就可以直接加4

2.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。

解：合理。ROM是只读存储器，作为指令存储器用来存储指令；DM用来向内存中存储数据，不需要特别快的速度，但需要足够的存储空间，RAM可以满足要求；GRF需要非常快的存储速度，上面三种存储器寄存器存取数据的速度最快。

1.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。）

解：RegDst=$\overline{o_5}\,\overline{o_4}\,\overline{o_3}\,\overline{o_2}\,\overline{o_1}\,\overline{o_0}f_5\overline{f_4}\,\overline{f_3}\,\overline{f_2}\,\overline{f_0}$

ALUSrc=$\overline{o_5}\,\overline{o_4}o_3\overline{o_2}\,\overline{o_1}\,\overline{o_0}+o_5\overline{o_4}\,\overline{o_3}\,\overline{o_2}\,\overline{o_1}\,\overline{o_0}+o_5\overline{o_4}\,\overline{o_3}\,\overline{o_2}o_1o_0$

$=\overline{o_5}\,\overline{o_4}o_3\overline{o_2}\,\overline{o_1}\,\overline{o_0}+o_5\overline{o_4}\,\overline{o_2}o_1o_0$

MemtoReg=$o_5\overline{o_4}\,\overline{o_3}\,\overline{o_2}\,\overline{o_1}\,\overline{o_0}$

RegWrite=$\overline{o_5}\,\overline{o_4}\,\overline{o_3}\,\overline{o_2}\,\overline{o_1}\,\overline{o_0}f_5\overline{f_4}\,\overline{f_3}\,\overline{f_2}\,\overline{f_0}+\overline{o_5}\,\overline{o_4}o_3\overline{o_2}\,\overline{o_1}\,\overline{o_0}+o_5\overline{o_4}\,\overline{o_3}\,\overline{o_2}o_1o_0$

nPC_sel=$o_5o_4o_3\overline{o_2}o_1o_0$

ExtOp=$o_5\overline{o_4}\,\overline{o_3}\,\overline{o_2}\,\overline{o_1}\,\overline{o_0}+o_5\overline{o_4}\,\overline{o_3}\,\overline{o_2}o_1o_0$

$=o_5\overline{o_4}\,\overline{o_2}o_1o_0$

2.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。

解：将X按照方便化简的原则当成0或1

RegDst=$\overline{o_5}\,\overline{o_4}\,\overline{o_3}\,\overline{o_2}\,\overline{o_1}\,\overline{o_0}f_5\overline{f_4}\,\overline{f_3}\,\overline{f_2}\,\overline{f_0}$

ALUSrc=$\overline{o_5}\overline{o_4}o_3o_2\overline{o_1}o_0+o_5\overline{o_4}\overline{o_3}\overline{o_2}\overline{o_1}o_0+o_5\overline{o_4}o_3\overline{o_2}o_1o_0$

$=\overline{o_5}\overline{o_4}o_3o_2\overline{o_1}o_0+o_5\overline{o_4}\overline{o_2}o_1o_0$

MemtoReg=$o_5\overline{o_4}o_3o_2o_1o_0$

RegWrite=$\overline{o_5}\overline{o_4}\overline{o_3}\overline{o_2}\overline{o_1}\overline{o_0}f_5\overline{f_4}\overline{f_3}\overline{f_2}f_0+\overline{o_5}\overline{o_4}o_3o_2\overline{o_1}o_0+o_5\overline{o_4}o_3o_2o_1o_0$

nPC_sel=$o_5o_4o_3\overline{o_2}o_1o_0$

ExtOp=$o_5\overline{o_4}o_3o_2o_1o_0+o_5\overline{o_4}o_3\overline{o_2}o_1o_0$

$=o_5\overline{o_4}o_2o_1o_0$

3.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。

解：nop 指令没对逻辑电路电路中的元件进行任何操作，它的存在与否对电路没有影响。

1.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。

解：假设 DM 有 256MB 容量， 并且映射在 0x3000_0000～0x3FFF_FFFF 区间。 那么只需要把高 4 位地址与 0x3 进行比较，比较结果就是 DM 的片选信号 。之前的 DM 存满后就从 0x3000_0000~0x3FFF_FFFF 存储数据。这次的 DM 最多存到 0x0000_0080，所以不需要片选信号。

2.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。

解：

形式验证的优点：

1.由于形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，因此测试者不必考虑如何获得测试向量。

2. 形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了测试验证的不足。

3. 形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，有可能缩短设计周期。

形式验证的缺点：

形式验证到目前为止仍然不能有效的验证电路的性能，如电路的时延和功耗等。