P3-Logisim 单周期 CPU 设计文档

一、设计说明

使用 Logisim 开发一个简单的 MIPS 单周期处理器，设计说明如下：

1. 处理器为 32 位处理器。
2. 处理器应支持的指令集为：{addu, subu, ori, lw, sw, beq, lui, nop}。
3. nop 机器码为 0x00000000。
4. addu,subu 可以不支持溢出。
5. 处理器为单周期设计。

二、模块规格

1. IFU

(1) 端口说明

表 1 IFU 端口说明

序号	信号	方向	描述
1	Clk	I	时钟信号
2	Reset	I	异步复位信号，将 PC 的值设置为 0x00000000 1：复位 0：无效
3	Branch	I	判断当前指令是否为 beq 指令 1：是 0：不是
4	Zero	I	判断 PC 是否满足 beq 指令跳转条件 1：满足 0：不满足
5	Offset [15:0]	I	16 位立即数偏移量
6	Instr [31:0]	O	32 位 IM 取出指令

(2) 功能定义

表 2 IFU 功能定义

序号	功能	描述
1	复位	Reset 有效时，PC 被设置为 0x00000000
2	计算 PC 的下一个值	当 Branch 信号和 Zero 信号均有效时 $PC = PC + 4 + signed(imm)$，否则 $PC = PC + 4$

3	指令输出	根据 PC 值取出 32 位指令

2. GRF

(1) 端口说明

表 3 GRF **端口说明**

序号	信号	方向	描述
1	Clk	I	时钟信号
2	Reset	I	异步复位信号，将 32 个寄存器中的值全部清零 1：复位 0：无效
3	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
4	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
5	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写入的目标寄存器
6	WD [31:0]	I	32 位写入数据
7	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
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 4 GRF **功能定义**

序号	功能	描述
1	复位	Reset 有效时，将 32 个寄存器中的值全部清零
2	读数据	读出 A1, A2 地址对应寄存器中所存储数据到 RD1, RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中

3. ALU

(1) 端口说明

表 5 ALU **端口说明**

序号	信号	方向	描述
1	A [31:0]	I	参与 ALU 计算的第一个 32 位数据
2	B [31:0]	I	参与 ALU 计算的第二个 32 位数据
3	ALUOpe [2:0]	I	ALU 功能的选择信号： 000：ALU 进行与运算

			001：ALU 进行或运算 010：ALU 进行加法运算 011：ALU 进行减法运算
4	C[31:0]	O	ALU 的计算结果

(2)功能定义

表 6 ALU 功能定义

序号	功能	描述
1	与运算	$C = A \,\&\, B$
2	或运算	$C = A \mid B$
3	加运算	$C = A + B$
4	减运算	$C = A - B$

4. DM

(1) 端口说明

表 7 DM 端口说明

序号	信号	方向	描述
1	Clk	I	时钟信号
2	Reset	I	异步复位信号，将 DM 中的值全部清零 1：复位 0：无效
3	MemRead	I	写使能信号 1：可向 DM 中写入数据 0：无效
4	MemWrite	I	读使能信号 1：可读取 DM 中数据 0：无效
5	Address [4:0]	I	5 位地址输入信号，对 DM 指定地址进行读写操作
6	WriteData [31:0]	I	32 位写入数据
7	ReadData [31:0]	O	32 位输出数据

(2)功能定义

表 8 DM 功能定义

序号	功能	描述
1	复位	Reset 有效时，将 DM 中的值全部清零
2	读操作	读出 Address 地址对应存储数据到 ReadData
3	写操作	当时钟上升沿来临时，将 WriteData 写入 Address 地址对应位置

5. EXT

(1) 端口说明

表 9 EXT 端口说明

序号	信号	方向	描述
1	Imm16 [15:0]	I	16 位待扩展数据
2	Imm32 [31:0]	O	32 位输出数据
3	EXTop	I	扩展操作信号 0：进行无符号扩展 1：进行符号扩展

(2)功能定义

表 10 EXT 功能定义

序号	功能	描述
1	无符号扩展	将 16 位立即数 Imm16 无符号拓展至 32 位输出 Imm32
2	符号扩展	将 16 位立即数 Imm16 符号拓展至 32 位输出 Imm32

6. Controller

见“三、控制器设计”章节

三、控制器设计

在设计单周期 CPU 时我参考了牛建伟老师的作业图——

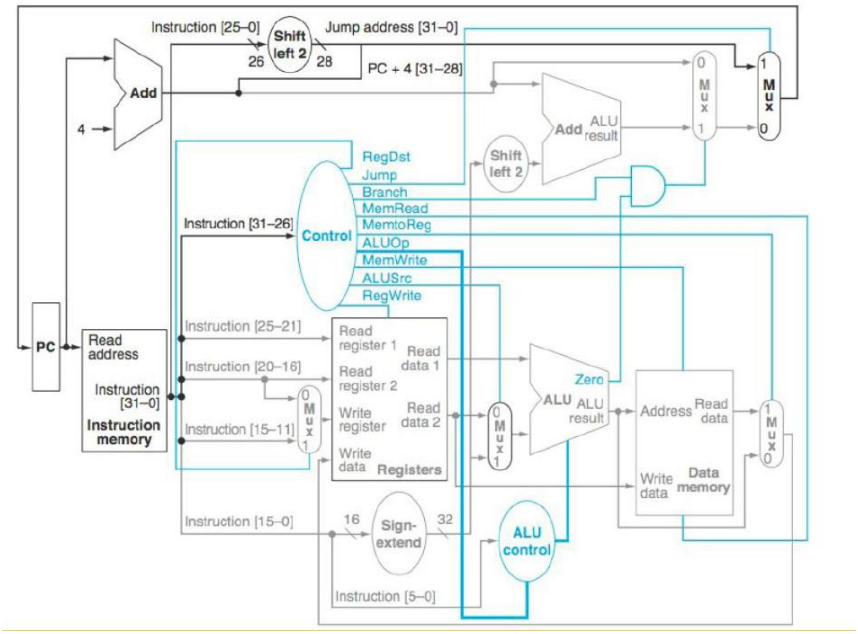

图 1 参考 CPU 电路图

将控制器部分分为 Control 模块和 ALU Control 模块，具体说明如下：

1. Control 模块端口与功能说明

表 11 Control 模块端口说明

序号	信号	方向	描述
1	Op [5:0]	I	6 位控制信号
2	RegDst	O	0：将 Rt 作为 GRF 的 A3 写入地址 1：将 Rd 作为 GRF 的 A3 写入地址
3	ALUSrc	O	0：将 RD2 作为参与 ALU 计算的第二个数据 1：将 EXT 输出作为参与 ALU 计算的第二个数据
4	MemToReg [1:0]	O	00：将 ALU 计算结果写入 GRF 01：将 DM 输出值写入 GRF 10：将填充到高位的 16 位立即数写入 GRF
5	RegWrite	O	0：GRF 写使能信号无效 1：GRF 写使能信号有效
6	MemRead	O	0：DM 读使能信号无效 1：DM 读使能信号有效
7	MemWrite	O	0：DM 写使能信号无效 1：DM 写使能信号有效
8	Branch	O	0：当前指令不是 beq 指令 1：当前指令为 beq 指令
9	ExtOp	O	0：进行无符号扩展 1：进行符号扩展
10	ALUOp [2:0]	O	000：R 型指令由功能码字段决定 001：lw/sw 指令所用加法 010：beq 指令所用减法 011：ori 指令所用或运算

2. Control 模块真值表

表 12 Control 模块真值表

Op 字段	000000	000000	001101	100011	101011	000100	001111
指令	addu(R)	subu(R)	ori	lw	sw	beq	lui
RegDst	1	1	0	0	0	0	0
ALUSrc	0	0	1	1	1	0	0
MemToReg [1:0]	00	00	00	01	00	00	10
RegWrite	1	1	1	1	0	0	1
MemRead	0	0	0	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	0

ExtOp	0	0	1	0	0	0	0
ALUOp [2:0]	000	000	111	001	001	010	011

3. ALU Control 模块真值表

表 13 ALU Control 模块真值表

指令	Func 字段	ALUOp	ALU 运算类型	ALUOpe
addu	100001	000	加	010
subu	100011	000	减	011
ori	xxxxxx	011	或	001
lw	xxxxxx	001	加	010
sw	xxxxxx	001	加	010
beq	xxxxxx	010	减	011
lui	xxxxxx	xxx	无	无
nop	xxxxxx	xxx	无	无

四、测试程序

1. 测试程序代码

```
# 测试程序代码:
.text
# ori 指令测试
    ori     $s0,$zero,15
    ori     $s1,$zero,31

# addu 指令测试
    addu    $s2,$s0,$s1
    addu    $s3,$s0,$s2

# subu 指令测试
    subu    $s4,$s3,$s2
    subu    $s5,$s2,$s1

# lui 指令测试
    lui     $s7,2333

# sw 指令测试
    sw  $s0,0($zero)
    sw  $s1,4($zero)

```

```
# lw 指令测试
    lw  $t0,0($zero)
    lw  $t1,4($zero)

# beq 指令测试
    beq     $s0,$s1,equal1
    ori $t2,$zero,1

equal1:
    beq     $s1,$s1,equal2
    ori $t3,$zero,2
equal2:
    ori $t4,$zero,666
```

2. 测试程序结果

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003000
hi		0x00000000
lo		0x00000000

图 2 测试程序结果(GRF)

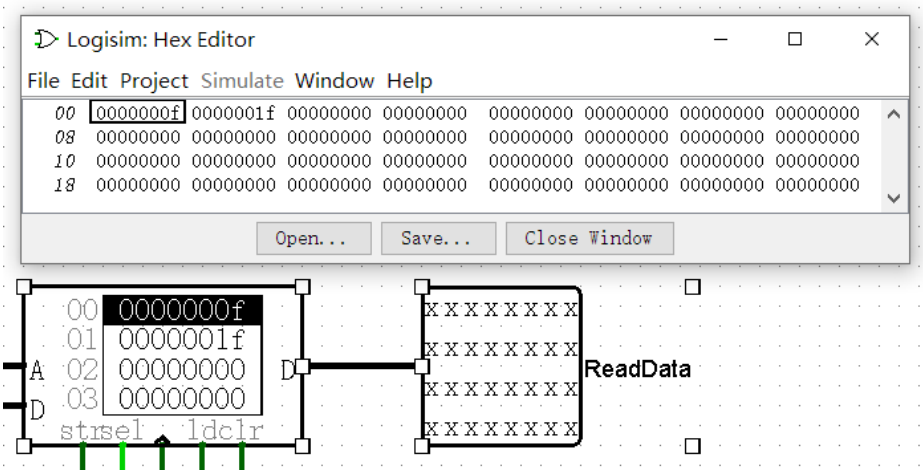

图 3 测试程序结果(DM)

五、思考题

1. 模块规格（L0.T2）

(1) 若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣。

优势：30 位 PC 所用硬件资源更少

劣势：30 位 PC 进行 jr 等指令时需要对 PC 进行扩展，增加了 CPU 设计的复杂度。

(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。

合理，分析如下：

① ROM 为只读存储器，而 IM 指令寄存器在 CPU 运行过程中只进行读取操作，故 IM 使用 ROM 合理。

② DM 需要进行读写操作，对速度要求不高。而 RAM 和寄存器都能实现读写操作，但 DM 对内存空间需求较大，若采用寄存器成本较高，因此选择读写速度较慢但存储空间大的 RAM 合理。

③ GRF 需要进行读写操作，使用频率高，对速度要求高。使用寄存器搭建 GRF 合理。

2. 控制器设计（L0.T3）

(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。）

$$RegDst = \overline{o5} \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot \overline{o1} \cdot f5 \cdot \overline{f4} \cdot \overline{f3} \cdot \overline{f2} \cdot \overline{f0}$$

$$ALUSrc = \overline{o5} \cdot \overline{o4} \cdot o3 \cdot o2 \cdot \overline{o1} \cdot o0 + o5 \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot o1 \cdot o0 + o5 \cdot \overline{o4} \cdot o3 \cdot \overline{o2} \cdot o1 \cdot o0$$

$$MemtoReg = o5 \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot o1 \cdot o0$$

$$RegWrite = \overline{o5} \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot \overline{o1} \cdot f5 \cdot \overline{f4} \cdot \overline{f3} \cdot \overline{f2} \cdot \overline{f0} + \overline{o5} \cdot \overline{o4} \cdot o3 \cdot o2 \cdot \overline{o1} \cdot o0 + o5 \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot o1 \cdot o0$$

$$nPC\_Sel = \overline{o5} \cdot \overline{o4} \cdot \overline{o3} \cdot o2 \cdot \overline{o1} \cdot \overline{o0}$$

$$ExtOp = o5 \cdot \overline{o4} \cdot \overline{o2} \cdot o1 \cdot o0$$

(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。

$$RegDst = \overline{o5} \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot \overline{o1} \cdot f5 \cdot \overline{f4} \cdot \overline{f3} \cdot \overline{f2} \cdot \overline{f0}$$

$$ALUSrc = \overline{o5} \cdot \overline{o4} \cdot o3 \cdot o2 \cdot \overline{o1} \cdot o0 + o5 \cdot \overline{o4} \cdot \overline{o2} \cdot o1 \cdot o0$$

$$MemtoReg = o5 \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot o1 \cdot o0$$

$$RegWrite = \overline{o5} \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot \overline{o1} \cdot f5 \cdot \overline{f4} \cdot \overline{f3} \cdot \overline{f2} \cdot \overline{f0} + \overline{o5} \cdot \overline{o4} \cdot o3 \cdot o2 \cdot \overline{o1} \cdot o0 + o5 \cdot \overline{o4} \cdot \overline{o3} \cdot \overline{o2} \cdot o1 \cdot o0$$

$$nPC\_Sel = \overline{o5} \cdot \overline{o4} \cdot \overline{o3} \cdot o2 \cdot \overline{o1} \cdot \overline{o0}$$

$$ExtOp = o5 \cdot \overline{o4} \cdot \overline{o2} \cdot o1 \cdot o0$$

(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。

nop 的指令为 0x00000000，仅进行 PC+4 操作，不对电路其它部分产生影响。

3. 测试 CPU（L0.T4）

(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。

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，结果就是 DM 的片选信号。[1]

(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。

形式验证的优点如下：

(1)形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了 100%。

(2)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，不需要开发测试激励。

(3)形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

形式验证的缺点如下：

形式验证只能检验电路设计的正确性，却无法检验其它方面如电路能耗等的优劣。[2]

[1] 参考百度文库《Project3_Logisim 完成单周期处理器开发》一文

[2] 参考百度百科“形式验证”词条