Logisim 单周期 CPU 设计文档

设计者：忽骁

一 CPU 设计文档

1 模块规格

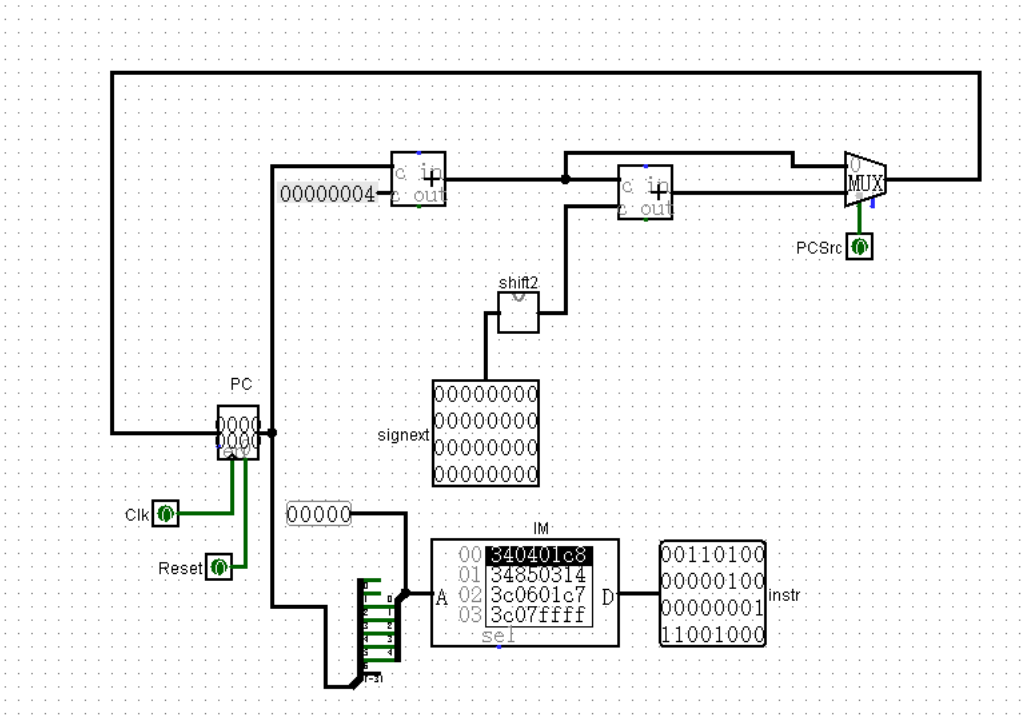
1）IFU（取指令单元）

PC（程序计数器）端口说明

序号	信号名称	位宽	方向	功能描述
1	Clk	1	I	时钟信号，控制 PC 新值的读入
2	Reset	1	I	复位信号，有效时将 PC 内的值置为 0x00000000
3	Data	32	I	运算出的新 PC 值，时钟上升沿时写入 PC 寄存器中
4	Output	32	O	当前 PC 值，包含当前指令的地址信息

IM(指令存储器)端口说明

序	信	位	方	功能描述
1	Addr	5	I	地址信号，根据此信号从 IM 中读出数据作为当前指令
2	Data	32	O	从 IM 中取出的指令



IFU 的内部具体实现

如上图，PC 输出的当前 PC 值经处理后作为 IM 的地址输入，IM 根据该地

址从相应的位置提取出数据作为指令，而 PC+4 与 PC+4+signext（imm6）<<2 由 PCSrc 信号选取出一个作为新 PC 值，在下一个上升沿到来时存入 PC 中，因为 PC 为 32 位，而 IM 的地址输入只有 5 位，所以选取 PC[6:2]作为地址输入

2）GRF（通用寄存器组）

GRF（通用寄存器组）端口说明

序号	信号名称	位宽	方向	功能描述
1	RegWrite	1	I	寄存器组写入信号，该信号有效时，寄存器组进行写操作
2	Clk	1	I	时钟信号，控制寄存器组的写入
3	Reset	1	I	复位信号，该信号有效时，寄存器组内所有寄存器被置 0
4	Reg1	5	I	寄存器读取端地址 1，根据此信号从相应寄存器中读取数据到 Rdata1
5	Reg2	5	I	寄存器读取端地址 2，根据此信号从相应寄存器中读取数据到 Rdata2
6	Wreg	5	I	寄存器写入端地址，时钟上升沿到来且 RegWrite 有效时，根据此信号将 Wdata 中数据在写入相应寄存器中
7	Wdata	32	I	寄存器写入数据，当时钟上升沿到来且 RegWrite 有效时，将 Wdata 中数据在写入相应寄存器中
8	Rdata1	32	O	寄存器读出的数据 1
9	Rdata2	32	O	寄存器读出的数据 2

3）ALU（算术逻辑单元）

ALU（算术逻辑单元）端口说明

序号	信号名称	位宽	方向	功能描述
1	ALU operation	4	I	ALU 功能选择信号，根据此信号进行相应的运算
2	A	32	I	第一个运算数
3	B	32	I	第二个运算数
4	Result	32	O	运算结果
5	Equal	1	O	相等判定信号，如果 A=B，则该信号为 1，否则为 0

ALU（算术逻辑单元）端口说明

序号	ALU Operation	运算描述	具体运算
1	0000	加法运算	A+B
2	0001	减法运算	A-B
3	0010	或运算	A|B
4	0011	乘法运算	A*B

4）DM（数据存储器）

DM（数据存储器）端口说明

序号	信号名称	位宽	方向	功能描述
1	Clk	1	I	时钟信号，控制 DM 的数据写入
2	Reset	1	I	复位信号，该信号有效时将 DM 内所有数据置为 0x00000000
3	MemRead	1	I	读取信号，该信号有效时 DM 读数据
4	MemWrite	1	I	写入信号，该信号有效时 DM 写数据
5	Addr	5	I	地址信号，根据此信号在相应位置执行写入或读取操作
6	Wdata	32	I	写入数据，写入操作时将其写入相应位置
7	Rdata	32	O	读出数据，读取操作时读出相应位置的数据

5）EXT

EXT 端口说明

序号	信号名称	位宽	方向	功能描述
1	Imm16	16	I	16 位立即数
2	extended	32	O	将 16 位立即数无符号扩展至 32 位后的结果

6） Controller（控制器）

Controller（控制器）端口说明

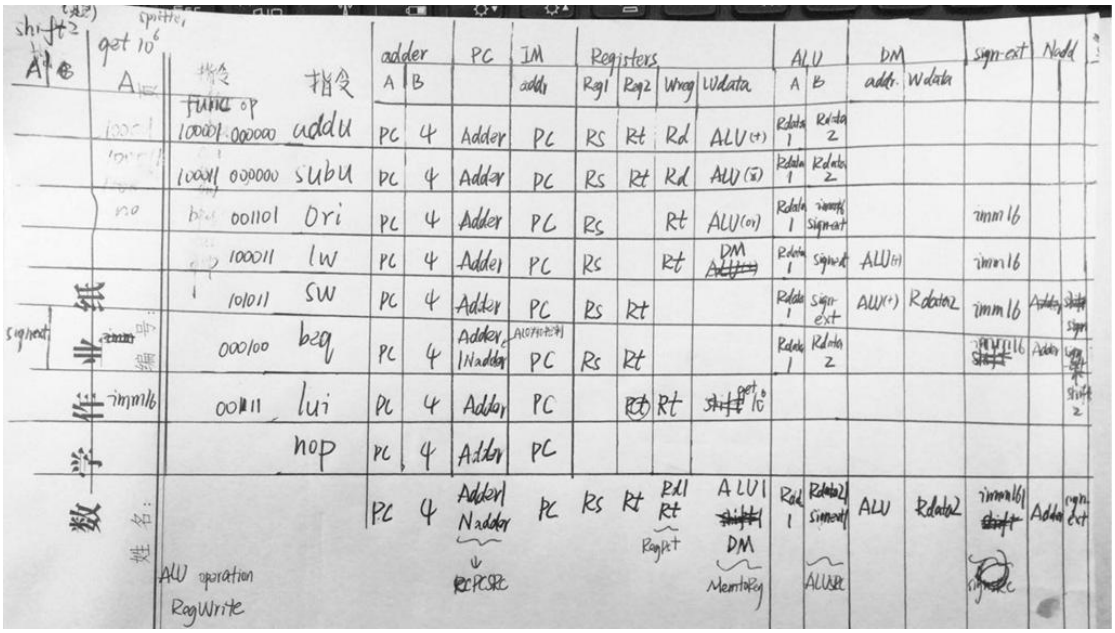
序号	信号名称	位宽	方向	功能描述
1	Op	6	I	指令的操作码字段（Op）
2	Func	6	I	指令的功能码字段（Func）
3	RegDst	1	O	寄存器写入端地址选择信号
4	ALUSrc	1	O	ALU 输入端 B 选择信号
5	MemtoReg	2	O	寄存器堆写入端数据选择信号
6	Branch	1	O	条件跳转信号（beq）判断信号

7	RegWrite	1	O	寄存器堆写入信号
8	MemRead	1	O	DM 数据读取信号
9	MemWrite	1	O	DM 数据写入信号
10	ALUOperation	4	O	ALU 功能选择信号

2 控制器设计

1）数据通路及其合并结果

首先可以得到数据通路及其合并结果如下



数据通路及其合并结果

2）主控单元信号

控制信号	失效时作用	有效时作用
RegDst	寄存器堆写入端地址来选择 Rt 字段	寄存器堆写入端地址选择 Rd 字段
RegWrite	无	把数据写入寄存器堆中对应寄存器
ALUSrc	ALU 输入端 B 选择寄存器堆输出 R[rt]	ALU 输入端 B 选择 Signext 输出
PCSrc	PC 输入源选择 PC+4	PC 输入选择 beq 指令的目的地址
MemRead	无	数据存储器 DM 读数据（输出）
MemWrite	无	数据存储器 DM 写数据（输入）

3）ALU 控制信号分析

ALU（算术逻辑单元）端口说明

序号	ALU operation	运算描述	具体运算
1	0000	加法运算	A+B
2	0001	减法运算	A-B
3	0010	或运算	A|B
4	0011	乘法运算	A*B

由数据通路我们可以得到：

①lw，sw 指令：ALU operation=0000，ALU 执行加法运算

②beq，subu 指令：ALU operation=0001，ALU 执行减法运算

③ori 指令：ALU operation=0010，ALU 执行或运算

4）主控单元信号分析

由此我们可以得到主控单元控制信号的分析如下：

①RegDst

R 型指令：RegDst=1，选择 Rd

Lw 指令： RegDst=0，选择 Rt

其他指令：不关心

②ALUSrc

R 型指令：ALUSrc=0，选择寄存器堆的 Read data2 输出

Beq 指令（减法运算）： ALUSrc=0，选择 Read data2 输出

Lw 指令，Sw 指令，Ori 指令： ALUSrc=1，选择 Signext 的输出

Sw 指令： ALUSrc=1，选择 Signext 的输出

③MemtoReg

R 型指令，Ori 指令：MemtoReg=00，选择 ALU 输出

Lw 指令： MemtoReg=01，选择数据存储器 DM 输出

Lui 指令：MemtoReg=10，选择 get10^6 移位扩展器输出

其他指令： 不关心 （不管选择哪个输出，RegWrite=0 就不会输出）

④Branch

Beq 指令：Branch=1，此时若 Zero=1，PC 输入选择加法器 Nadd 输出（分支指令目的地址），否则选择加法器 Add 输出（PC+4）

其他指令：Branch=0，PC 输入选择加法器 Add 输出（PC+4）

5）读写控制信号分析

①RegWrite

R 类指令，Ori、Lw、Lui 指令：Regwrite=1，寄存器堆执行写操作

其他指令：RegWrite=0，寄存器堆不执行写操作

②MemRead

Lw 指令：MemRead=1，DM 执行读出操作

其他指令：MemRead=0，DM 不执行读出操作

③MemWrite

Sw 指令：MemWrite=1，DM 执行写入操作

其他指令，MemWrite=0，Dm 不执行写入操作

6）控制信号真值表

控制信号真值表

Func:	100001	100011	None				
Op:	00000	00000	001101	100011	101011	000100	001111
	addu	subu	ori	lw	sw	beq	lui
RegDst	**1**	**1**	**0**	**0**	**X**	**X**	**0**
RegWrite	**1**	**1**	**1**	**1**	**0**	**0**	**1**
ALUSrc	**0**	**0**	**1**	**1**	**1**	**0**	**X**
Branch	**0**	**0**	**0**	**0**	**0**	**1**	**0**
MemRead	**0**	**0**	**0**	**1**	**0**	**0**	**0**
MemWrite	**0**	**0**	**0**	**0**	**1**	**0**	**0**
MemtoReg<1:0>	**00**	**00**	**00**	**01**	**XX**	**XX**	**10**
ALU operation<3:0>	**0000**	**0001**	**0010**	**0000**	**0000**	**0001**	**XXXX**

注：其中 branch 与 ALU 的相等判断端同时为 1 时，新 PC 值选择信号 PCSrc=1（进行与运算）

3 测试程序

1）测试程序

```
    .text
#test ori
    ori $a0,$0,456
    ori $a1,$a0,788
#test lui
    lui $a2,455
```

```
   lui $a3,0xffff #sign
   ori $a3,$a3,0xffff #set $a3 to -1
#twst addu
   addu $s0,$a0,$0
   addu $s1,$a0,$a1
   addu $s2,$0,$0
#twst subu
   subu $s3,$0,$0
   subu $s4,$0,$a0
   subu $s5,$a0,$0
   subu $s6,$a1,$a0
#test sw
   ori $t0,0x0000
   sw $a0,0($t0)
   sw $a1,4($t0)
   sw $a2,8($t0)
   sw $a3,12($t0)
   sw $s4,16($t0)
```

```
   sw $s5,20($t0)
#test lw
   lw $a1,0($t0)
   lw $a0,4($t0)
   lw $a3,8($t0)
   lw $a2,12($t0)
   lw $s5,16($t0)
   lw $s4,20($t0)
#test beq
   ori $a0,$0,1
   ori $a1,$0,2
   ori $a2,$0,1
   beq $a0,$a1,loop1
   beq $a0,$a2,loop2
loop1:
   sw $a0,24($t0)
loop2:
   sw $a1,28($t1)
```

2）测试期望

如果 CPU 运行正确，则运行完所有指令后我们会得到寄存器中的值如下

寄存器存储情况

	$4 ($a0)	$5 ($a1)	$6 ($a2)	$7 ($a3)	$16 ($s0)	$17 ($s1)	$20 ($s4)	$21 ($s5)	$22 ($s6)
Value	0x00000001	0x00000002	0x00000001	0x01c70000	0x000001c8	0x000005a4	0x000001c8	0xfffffe38	0x00000214

DM 中存储情况如下

DM 存储情况

Addr（0x00000000）	Addr+0	Addr+4	Addr+8	Addr+c	Addr+10	Addr+14	Addr+18	Addr+28
value	0x000001c8	0x000003dc	0x01c70000	0xffffffff	0xfffffe38	0x000001c8	0x00000000	0x00000002

注：因为第二条 beq 指令的条件满足，直接跳过倒数第二条存储指令，因此开始地址为 0x00000018 的空间内存储的数为 0

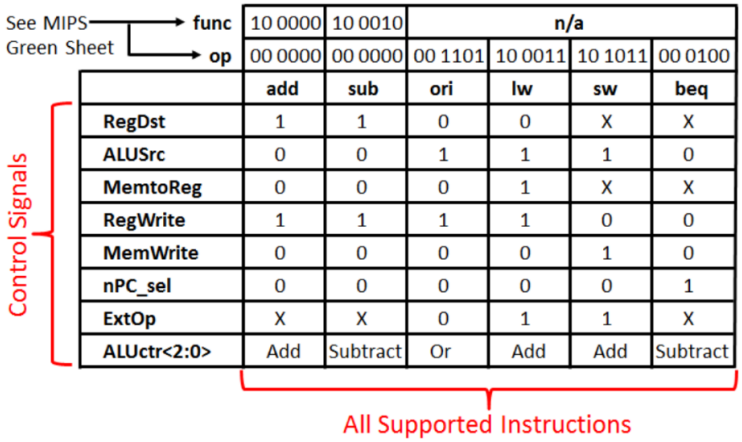
二 思考题

1）模块规格部分

1 因为 mips 中的指令都 32bit 的，所以其存储空间都是以四字节为单位的，因此 PC 必定为 4 的倍数，即 PC 的最低两位必为 0，若采用 30 位的 PC，即将最低两位舍去，简化了 PC 的形式，但是却需要改动得到新 PC 值的运算过程，所以一定程度上增添了麻烦。而 32bit 的 PC 地址形式不如 30bit 的 PC 的简洁，但是新 PC 值的运算可以沿用之前的设计。

2 我认为是合理的，因为 IM 只需要根据 PC 值读出当前指令，不需要通过端口写入数据，也不需要时钟控制以及通过端口复位，因此 ROM 已经足够满足 IM 所需的功能。而 DM 需要时钟控制，还需要通过端口控制读取数据，写入数据以及置 0，而 Separate load and store ports 的 RAM 就提供了相应的端口，能够满足 DM 所需要的功能。而对于 GRF，它需要时钟同步所有寄存器，并且保存写入的数据，在写入信号有效时写入相应的数据，以及读出相应寄存器内的数据，因此寄存器能够满足 GRF 所需要的功能。

2）控制器设计部分



	add	sub	ori	lw	sw	beq
func	10 0000	10 0010	n/a			
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100
RegDst	1	1	0	0	X	X
ALUSrc	0	0	1	1	1	0
MemtoReg	0	0	0	1	X	X
RegWrite	1	1	1	1	0	0
MemWrite	0	0	0	0	1	0
nPC_sel	0	0	0	0	0	1
ExtOp	X	X	0	1	1	X
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract

题图

首先我们可以得到

add=(func5) ~(func4) ~(func3) ~(func2) ~(func1) ~(func0) ~(op5) ~(op4) ~(op3) ~(op2) ~(op1) ~(op0)

sub=(func5) ~(func4) ~(func3) ~(func2) (func1) ~(func0) ~(op5) ~(op4) ~(op3) ~(op2) ~(op1) ~(op0)

ori= ~(op5) ~(op4) (op3) (op2) ~(op1) (op0)

lw= (op5) ~(op4) ~(op3) ~(op2) (op1) (op0)

sw= (op5) ~(op4) (op3) ~(op2) (op1) (op0)

beq= ~(op5) ~(op4) ~(op3) (op2) ~(op1) ~(op0)

由此有 RegDst=(add)+(sub)

={(func5) ~(func4) ~(func3) ~(func2) ~(func1) ~(func0) ~(op5) ~(op4) ~(op3) ~(op2) ~(op1) ~(op0)}+{ (func5) ~(func4) ~(func3) ~(func2) (func1) ~(func0) ~(op5) ~(op4) ~(op3) ~(op2) ~(op1) ~(op0)}

ALUSrc=(ori)+(lw)+(sw)

={ ~(op5) ~(op4) (op3) (op2) ~(op1) (op0)}+{ (op5) ~(op4) ~(op3) ~(op2) (op1) (op0)}+{ (op5) ~(op4) (op3) ~(op2) (op1) (op0)}

MemtoReg=lw= (op5) ~(op4) ~(op3) ~(op2) (op1) (op0)

RegWrite=(add)+(sub)+(ori)+(lw)

={(func5) ~(func4) ~(func3) ~(func2) ~(func1) ~(func0) ~(op5) ~(op4) ~(op3) ~(op2) ~(op1) ~(op0)}+{ (func5) ~(func4) ~(func3) ~(func2) (func1) ~(func0) ~(op5) ~(op4) ~(op3) ~(op2) ~(op1) ~(op0)}+{ ~(op5) ~(op4) (op3) (op2) ~(op1) (op0)}+{ (op5) ~(op4) ~(op3) ~(op2) (op1) (op0)}

nPC_Sel=beq= ~(op5) ~(op4) ~(op3) (op2) ~(op1) ~(op0)

ExtOp=(lw)+(sw)

={ (op5) ~(op4) ~(op3) ~(op2) (op1) (op0)}+{ (op5) ~(op4) (op3) ~(op2) (op1) (op0)}

2

RegDst=~op0

ALUSrc=op0

MemtoReg=op1+op0

RegWrite=~{{(op5) ~(op4) (op3) ~(op2) (op1) (op0)}+{~(op5) ~(op4) (op3) (op2) ~(op1) ~(op0)}}={ ~(op5) |(op4) |~(op3) |(op2) |~(op1) |~(op0)) } {(op5) (op4) ~(op3) ~(op2) (op1) (op0)}

nPC_Sel=(op2) ~(op0)

ExtOp=op1&op0

3

因为 nop 指令码全为 0，且其不需要执行任何的操作，因此无需产生任何的控制信号，因此设计控制器时不需要考虑 nop

3）测试 CPU 部分

a.模拟方法严重依赖于测试向量的选取，合理而充分地选取测试向量本身并不容易，并且其效率不高。VLSI 的设计常常需要反复修改，而每次修改后都需要模拟，以确认本次修改达到了预期目的且没有引入新的错误

b.首先，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。其次，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，因此有效地克服了模拟验证的不足。 此外，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 然而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。因此，两种验证方式各有优缺点，相互配合使用才能达到最佳效果。