P3 设计文档

——Logisim 单周期

一、指令集

1. ADDU：不支持溢出的加法

编码	Opcode 000000	rs	rt	rd	Shamt 00000	Func 100001
	6	5	5	5	5	6
描述	GPR[rd] ← GPR[rs] + GPR[rt]					

2. SUBU：不支持溢出的减法

编码	Opcode 000000	rs	rt	rd	Shamt 00000	Func 100011
	6	5	5	5	5	6
描述	GPR[rd] ← GPR[rs] - GPR[rt]					

3. ORI：或立即数

编码	Opcode 001101	rs	rt	immediate
	6	5	5	16
描述	GPR[rt] ← GPR[rs] OR zero_extend(immediate)			

4. LW：加载字

编码	Opcode 100011	base	rt	offset
	6	5	5	16
描述	Addr ← GPR[base] + sign_extend(offset) GPR[rt] ← memory[Addr]			

5. SW：存储字

编码	Opcode 101011	base	rt	offset
	6	5	5	16
描述	Addr ← GPR[base] + sign_extend(offset) memory[Addr] ← GPR[rt]			

6. BEQ：相等时跳转

编码	Opcode 000100	rs	rt	offset
	6	5	5	16

描述	If (GPR[rs] == GPR[rt])
	PC ← PC + 4 + sign_extend(offset||0^2)
	Else
	PC ← PC + 4			

7. LUI：立即数加载至高位

编码	Opcode 001111	0 00000	rt	immediate
	6	5	5	16
描述	GPR[rt] ← immediate||0^16			

二、模块规格

1. IFU：取指令单元

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	异步复位信号
NPCIMM	[31:0]	I	计算 NPC 的立即数信号
NPCOp		I	计算 NPC 的选择信号
IM	[31:0]	O	当前指令信号

模块功能说明如下：

序号	功能名	描述
1	计算地址	计算下一条指令在 IM（指令存储器）中的地址
2	取指令	从 IM（指令存储器）中取出 32 位指令并通过 IM 端口输出
3	复位	reset 信号为 1 时，复位 PC（指令地址）至初始状态

2. GRF： 寄存器堆

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	异步复位信号
WE		I	写使能信号
A1	[4:0]	I	第一个地址输入信号
A2	[4:0]	I	第二个地址输入信号
A3	[4:0]	I	第三个地址输入信号
WD	[31:0]	I	写入数据信号
RD1	[31:0]	O	A1 所对应的寄存器的数据信号
RD2	[31:0]	O	A2 所对应的的寄存器的数据信号

模块功能说明如下：

序号	功能名	描述
1	读数据	读出 A1，A2 地址对应寄存器数据并通过 RD1，RD2 端口输出
2	写数据	Clk 上升沿到来时，若 WE 信号为 1，则向 A3 地址对应寄存器写入数据 WD（0 号寄存器不能被写入）
3	复位	reset 信号为 1 时，复位所有寄存器至初始状态

3. ALU：计算模块

模块端口定义如下：

信号名	位数	方向	描述
ALUA	[31:0]	I	参与运算的第一个数据信号
ALUB	[31:0]	I	参与运算的第二个数据信号
ALUOp	[2:0]	I	运算方式的选择信号
ALU	[31:0]	O	运算结果的数据信号

模块功能说明如下：

序号	功能名	描述
1	不支持溢出加法	若 ALUOp==001，计算 A+B 并通过 ALU 端口输出
2	不支持溢出减法	若 ALUOp==010，计算 A-B 并通过 ALU 端口输出
3	按位或运算	若 ALUOp==011，计算 A|B 并通过 ALU 端口输出

4. EXT：立即数拓展模块

模块端口定义如下：

信号名	位数	方向	描述
IMM	[15:0]	I	参与拓展的立即数信号
EXTOp	[2:0]	I	拓展方式的选择信号
EXT	[31:0]	O	拓展完成后的数据信号

模块功能说明如下：

序号	功能名	描述
1	零拓展	若 EXTOp==001，将立即数加载至输出信号低位并用 0 填充输出信号的高 16 位
2	符号拓展	若 EXTOp==010，将立即数加载至输出信号低位并用其最高位填充输出信号的高 16 位
3	补 00 符号拓展	若 EXTOp==011，将立即数末尾补两个 0 后进行符号拓展
4	加载至高位	若 EXTOp==100，将立即数加载至输出信号高位并用 0 填充输出信号的低 16 位

5. DM：数据存储器

模块端口定义如下：

信号名	位数	方向	描述
clk		I	内置时钟信号
reset		I	异步复位信号
DMA	[31:0]	I	存取的地址信号
DMD	[31:0]	I	存取的数据信号
DMOp		I	存数据使能信号
DM	[31:0]	O	取出的数据信号

模块功能说明如下：

序号	功能名	描述
1	存数据	clk 上升沿到来时，若 DMOp 信号为 1，则向 DMA 对应的地址中写入 DMD 数据
2	取数据	将 DMA 地址对应的数据通过 DM 端口输出
3	复位	reset 信号为 1 时，复位 DM（数据存储器）至初始状态

三、控制器设计

1. 控制信号说明

序号	信号名	位数	描述
1	NPCOp		作为计算 NPC 时的选择信号
2	GRFWE		作为 GRF 的写使能信号
3	ALUOp	[2:0]	作为 ALU 的计算方式选择信号
4	EXTOp	[2:0]	作为 EXT 的拓展方式选择信号
5	DMOp		作为 DM 的写使能信号
6	GRFA3_MUXOp		作为 GRF 的 A3 端口输入信号的选择信号
7	GRFWD_MUXOp	[1:0]	作为 GRF 的 WD 端口输入信号的选择信号
8	ALUB_MUXOp		作为 ALU 的 B 端口输入信号的选择信号

2. 指令与控制信号真值表

func	100001	100011						000000
opcode	000000	000000	001101	100011	101011	000100	001111	000000
指令名	addu	subu	ori	lw	sw	beq	lui	nop
NPCop	0	0	0	0	0	1	0	0
GRFWE	1	1	1	1	0	0	1	0
ALUOp	001	010	011	001	001	010	000	000
EXTOp	000	000	001	010	010	011	100	000
DMOp	0	0	0	0	1	0	0	0
GRFA3_MUXOp	1	1	0	0	0	0	0	0
GRFWD_MUXOp	00	00	00	01	00	00	10	00
ALUB_MUXOp	1	1	0	0	0	1	0	0

四、数据通路

1. 信号连接表

	IFU	GRF			
输入信号	NPCIMM	A1	A2	A3	WD
addu		IM.IM[25:21]	IM.IM[20:16]	IM.IM[15:11]	ALU.ALU
subu		IM.IM[25:21]	IM.IM[20:16]	IM.IM[15:11]	ALU.ALU
ori		IM.IM[25:21]		IM.IM[20:16]	ALU.ALU
lw		IM.IM[25:21]		IM.IM[20:16]	DM.DM
sw		IM.IM[25:21]	IM.IM[20:16]		
beq	EXT.EXT	IM.IM[25:21]	IM.IM[20:16]		
lui				IM.IM[20:16]	EXT.EXT
nop					
ALL	EXT.EXT	IM.IM[25:21]	IM.IM[20:16]	IM.IM[15:11] IM.IM[20:16]	ALU.ALU DM.DM EXT.EXT
	EXT	ALU		DM	
输入信号	EXTIMM	ALUA	ALUB	DMA	DMD
addu		GRF.RD1	GRF.RD2		
subu		GRF.RD1	GRF.RD2		
ori	IM.IM[15:0]	GRF.RD1	EXT.EXT		
lw	IM.IM[15:0]	GRF.RD1	EXT.EXT	ALU.ALU	
sw	IM.IM[15:0]	GRF.RD1	EXT.EXT	ALU.ALU	GRF.RD2
beq		GRF.RD1	GRF.RD2		
lui	IM.IM[15:0]	GRF.RD1			
nop					
ALL	IM.IM[15:0]	GRF.RD1	GRF.RD2 EXT.EXT	ALU.ALU	GRF.RD2

2. 冲突选择表

GRFA3				
选择信号 GRFA3_MUXOp	0		1	
连接信号	IM.IM[20:16]		IM.IM[15:11]	
GRFWD				
选择信号 GRFWD_MUXOp	00	01	10	11
连接信号	ALU.ALU	DM.DM	EXT.EXT	
ALUB				
选择信号 ALUB_MUXOp	0		1	
连接信号	EXT.EXT		RD2.RD2	

五、测试样例

```
ori $t1,$0,1            # t1 = 0x00000001
lui $t2,1               # t2 = 0x00010000
ori $t3,$0,0xffff       # t3 = 0x0000ffff
lui $t4,0xffff          # t4 = 0xffff0000
beq $t1,$t2,end
nop
addu $s1,$t1,$t2        # s1 = 0x00010001
addu $s2,$t2,$t4        # s2 = 0x00000000
subu $s3,$t2,$t1        # s3 = 0x0000ffff
subu $s4,$t1,$t3        # s4 = 0xffff0002
ori $t6,$0,4            # t6 = 0x00000004
sw $s3,0($t6)           # dm[4] = 0x0000ffff
sw $s4,4($t6)           # dm[8] = 0xffff0002
ori $t5,$0,8            # t5 = 0x00000008
lw $s5,0($t5)           # s5 = 0xffff0002
addu $t7,$t6,$0         # t7 = 0x00000004
addu $t8,$t1,$0         # t8 = 0x00000001
begin:
addu $t7,$t7,$t1        # t7 = t1 +1
beq $t7,$t5,end         # if (t7 = 0x00000008) jump to end
addu $t8,$t8,$t8        # t8 = t8 + t8
beq $0,$0,begin         # jump to begin
end:
nop
```

MARS 模拟结果：

Data Segment

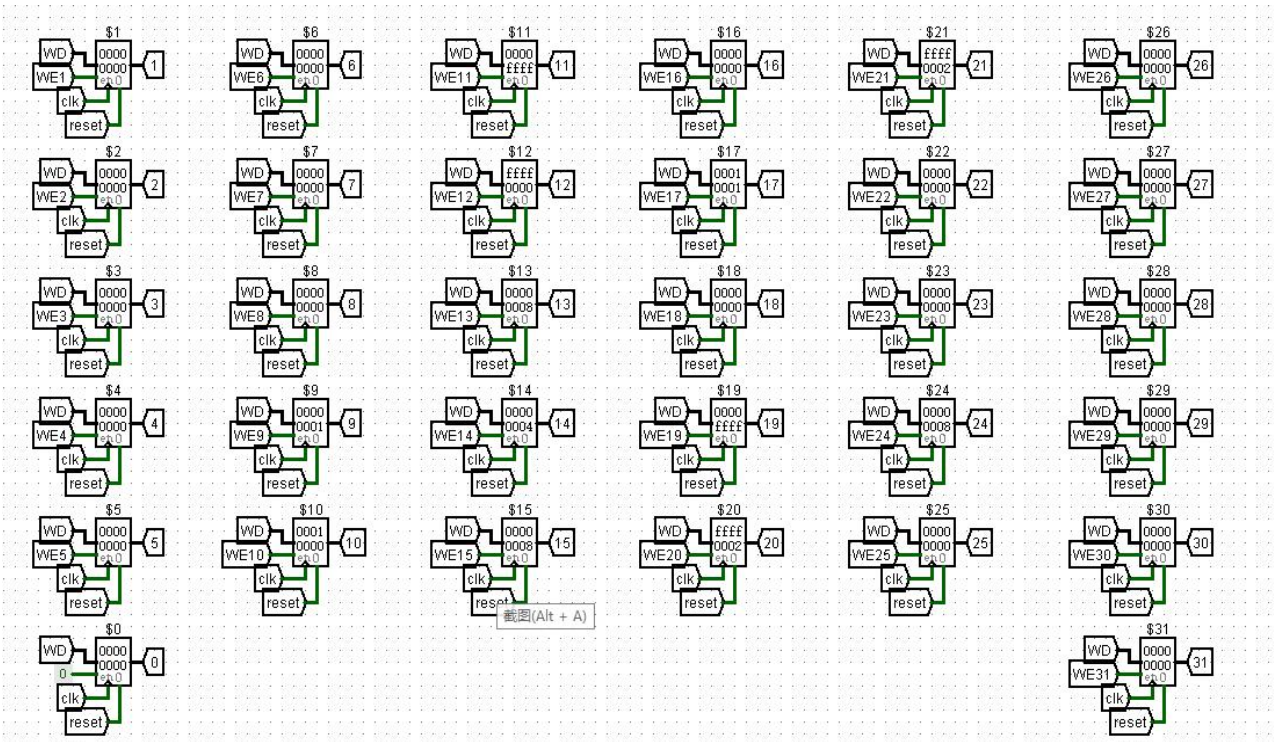
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x0000ffff	0xffff0002	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000001a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x00000000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

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	0x00000001
$t2	10	0x00010000
$t3	11	0x0000ffff
$t4	12	0xffff0000
$t5	13	0x00000008
$t6	14	0x00000004
$t7	15	0x00000008
$s0	16	0x00000000
$s1	17	0x00010001
$s2	18	0x00000000
$s3	19	0x0000ffff
$s4	20	0xffff0002
$s5	21	0xffff0002
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000008
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003058
hi		0x00000000
lo		0x00000000

本地运行结果：



00	00000000	0000ffff	ffff0002	00000000		00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000
10	00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000		00000000	00000000	00000000	00000000

思考题

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

32 位 PC 在进行 NPC 的计算时不需要进行拓展，但是其最低两位在系统中没有作用，造成一定的浪费。

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。

这种做法是合理的，IM 作为指令存储器，不需要对其进行写操作，适合用只读的 ROM 实现，DM 作为数据存储器，需要进行读写操作且存储空间要足够大，适合用双端口且存储空间大的 RAM 实现，GRF 作为数据临时存储器，需要存取速度快，适合用存取速度快的寄存器实现。

3. 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）

```
add=!op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&!func1&&!func2&&!
func3&&!func4&&func5
sub= !op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&func1&&!func2&&!f
unc3&&!func4  &&func5
ori=op0&&!op1&&op2&&op3&&!op4&&!op5
lw=op0&&op1&&!op2&&!op3&&!op4&&op5
sw=op0&&op1&&!op2&&op3&&!op4&&op5
beq=!op0&&!op1&&op2&&!op3&&!op4&&!op5

RegDst=add||sub
ALUSrc=ori||lw||sw
MemtoReg=lw
RegWrite=add||sub||ori||lw
nPC_Sel=beq
EXTOp=lw||sw
```

4. 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=!op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&!func2&&!func3
&&!func4&&func5
ALUSrc=(op0&&!op1&&op2&&op3&&!op4&&!op5)||(op0&&op1&&!op2&&!op4&&
op5)
MemtoReg=op0&&op1&&!op2&&!op3&&!op4&&op5
RegWrite=(!op0&&!op1&&!op2&&!op3&&!op4&&!op5&&!func0&&!func2&&!fu
nc3&&!func4&&func5)||(op0&&!op1&&op2&&op3&&!op4&&!op5)||(op0&&op1
&&!op2&&!op3&&!op4&&op5)
```

```
nPC_Sel=!op0&&!op1&&op2&&!op3&&!op4&&!op5
EXTOp=op0&&op1&&!op2&&!op4&&op5
```

5. 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 指令机器码为 0x00000000，无论将其视作何种指令，其 Opcode，rs，rt，rd，Shamt，func，immediate/offset 段全为 0，不会对 DM，GRF 等部件造成实际影响，因此不需要考虑 nop 指令对控制信号的控制。

6. 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。

若有多个片中能够存储数据，则需要将计算或拓展出的地址进行片选，具体方法是将所得地址与各片地址作比较，得出其所在的片段并做相应运算后存入所对应的位置。

7. 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。

与测试相比，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。

其缺点为：

（1）形式验证到目前为止仍然不能有效地验证电路的性能，如电路的时延与功效等。