

P3 设计文档

P3 设计文档

设计草稿

- 一、设计与测试说明
- 二、整体结构
 - 1.模块规格
 - 2.顶层电路外观
 - 3.与或门阵列（控制器实现）

思考题

- 整体结构
- 模块规格
- 控制器设计
- 测试 CPU

设计草稿

一、设计与测试说明

- 1. 处理器为 32 位单周期处理器
- 2. 采用 MIPS-C0 指令集，即{add（不支持溢出，事实上为addu），sub（不支持溢出，事实上为subu），ori, lw, sw, beq, lui, nop}
- 3. 采用模块化和层次化设计，顶层有效驱动信号仅为异步复位信号 reset，时钟信号采用内置时钟模块

二、整体结构

1.模块规格

- IFU
 - 端口说明

序号	信号名	方向	描述
1	clk	I	内置时钟信号
2	reset	I	异步复位信号 1'b1: 清零 1'b0: 保持
3	Zero	I	ALU 计算结果是否为 0 的标志: 1'b1: 计算结果为0 1'b0: 计算结果不为0

序号	信号名	方向	描述
4	NPCControl[1:0]	I	NPC 计算的控制信号, 接收 Controller 产生的 NPCControl 信号
5	PC[31:0]	O	32 位当前指令计数, 接收 NPC 的 NPC 信号
6	Instr[31:0]	O	32 位 MIPS 指令

- 功能定义

序号	功能	描述
1	异步复位	reset 信号为 1'b1 时, 清空寄存器
2	取指令	根据 PC 的值从 IM 中取出指令
3	输出 PC	输出当前指令计数

- NPC

- 端口说明

序号	信号名	方向	描述
1	reset	I	异步复位信号 1'b1: 清零 1'b0: 保持
2	NPCControl[1:0]	I	接收 Controller 的 NPCControl 信号
3	Zero	I	接收 ALU 的 Zero 信号
4	PC[31:0]	I	接收 IFU 的 PC 信号
5	Imm26[25:0]	I	接收 Splitter 的 Imm26 信号
6	EXTImm32[31:0]	I	接收 EXT 的 EXTImm32 信号
7	NPC[31:0]	O	Next PC, 下一条指令计数

- 功能定义

序号	功能	描述
1	异步复位	reset 信号为 1 时, 置 PC 初始值为 0x00000000
2	计算 NPC	<p>根据 NPCControl 计算 NPC</p> <p>NPCControl 为 2'b00 时: 递增, $NPC = PC + 4$</p> <p>NPCControl 为 2'b01 时: 计算 beq 指令地址: $NPC = PC + 4 + sign_extend(offset 0^2)$</p> <p>NPCControl 为 2'b10 时: 计算 j 指令地址: $NPC = PC[31:28] instr_index 0^2$</p>

- GRF
 - 端口说明

序号	信号名	方向	描述
1	clk	I	内置时钟信号
2	reset	I	异步复位信号 1'b1: 清零 1'b0: 保持
3	WE	I	写使能信号, 接受 RegWrite 1'b1: 可写入数据 1'b0: 不可写入数据
4	A1[4:0]	I	5 位地址位选信号, 接收Instr[25:21], 选择对应寄存器, 将其值输出到 RD1
5	A2[4:0]	I	5 位地址位选信号, 接收Instr[20:16], 选择对应寄存器, 将其值输出到 RD2
6	A3[4:0]	I	5 位地址位选信号, 选择寄存器输入 WD
7	WD[31:0]	I	32 位写入数据
8	RD1[31:0]	O	输出 A1 选择的寄存器所存储的值
9	RD2[31:0]	O	输出 A2 选择的寄存器所存储的值

- 功能定义

序号	功能	描述
1	异步复位	reset 信号为 1'b1 时, 清空所有寄存器
2	读寄存器	将 A1 和 A2 对应的寄存器中的值输出到 RD1 和 RD2
3	写寄存器	WE 为 1'b1 时, 在时钟上升沿将 WD 的值写入到 A3 对应的寄存器中

- ALU
 - 端口说明

序号	信号名	方向	描述
----	-----	----	----

序号	信号名	方向	描述
1	SrcA[31:0]	I	第一个运算数
2	SrcB[31:0]	I	第二个运算数
3	ALUControl[2:0]	I	ALU 控制信号，对应的操作为： 3'b000: + 3'b001: - 3'b010: & (按位) 3'b011: (按位)
4	Zero	O	SrcA 与 SrcB 是否相等的标志信号： 1: 相等 0: 不相等
5	ALUResult[31:0]	O	SrcA 与 SrcB 运算结果

◦ 功能定义

序号	功能	描述
1	加法	输出 SrcA + SrcB 到 ALUResult
2	减法	输出 SrcA - SrcB 到 ALUResult
3	按位与	输出 SrcA & SrcB 到 ALUResult
4	按位或	输出 SrcA SrcB 到 ALUResult
5	判断相等	输出 Zero 信号到 Zero

• Controller

◦ 端口说明

序号	信号名	方向	描述
1	opcode[5:0]	I	32 位 MIPS 指令中的 opcode 字段
2	funct[5:0]	I	32 位 MIPS 指令中的 funct 字段
3	ALUControl[2:0]	O	ALU 控制信号，确定 ALU 执行的功能，端口说明见 ALU
4	MemRead	O	DM 读信号
5	MemWrite	O	DM 写信号
6	RegWrite	O	GRF 写信号

序号	信号名	方向	描述
7	Mem2Reg[1:0]	O	GRF 写入数据的选择信号 2'b00: ALU 运算结果 2'b01: Memory[Addr] 2'b10: EXT 运算结果
8	EXTControl[1:0]	O	EXT 扩展方式选择信号 2'b00: 零扩展 2'b01: 符号扩展 2'b10: 低 16 位零扩展
9	ALUSrc	O	ALU 的第二个操作数选择信号 1'b0: RD2 1'b1: EXT 扩展的立即数
10	RegDst	O	寄存器写地址控制 1'b0: rt 1'b1: rd
11	NPCControl[1:0]	O	NPC 计算方式选择信号 2'b00: 递增 2'b01: beq 指令跳转 2'b10: j 指令跳转

- 功能定义

序号	功能	描述
1	产生 ALU 控制信号	-
2	产生 GRF 控制信号	-
3	产生 DM 控制信号	-
4	产生 EXT 控制信号	-
5	产生 IFU 控制信号	-

- EXT

- 端口说明

序号	信号名	方向	描述
1	Imm16[15:0]	I	16 位需要扩展的立即数
2	EXTControl[1:0]	I	符号扩展的标志信号 2'b00: 零扩展 2'b01: 符号扩展 2'b10: 低位零扩展
3	EXTResult[31:0]	O	32 位扩展结果

◦ 功能定义

序号	功能	描述
1	高位符号扩展	-
2	高位零扩展	-
3	低位零扩展	-

• DM

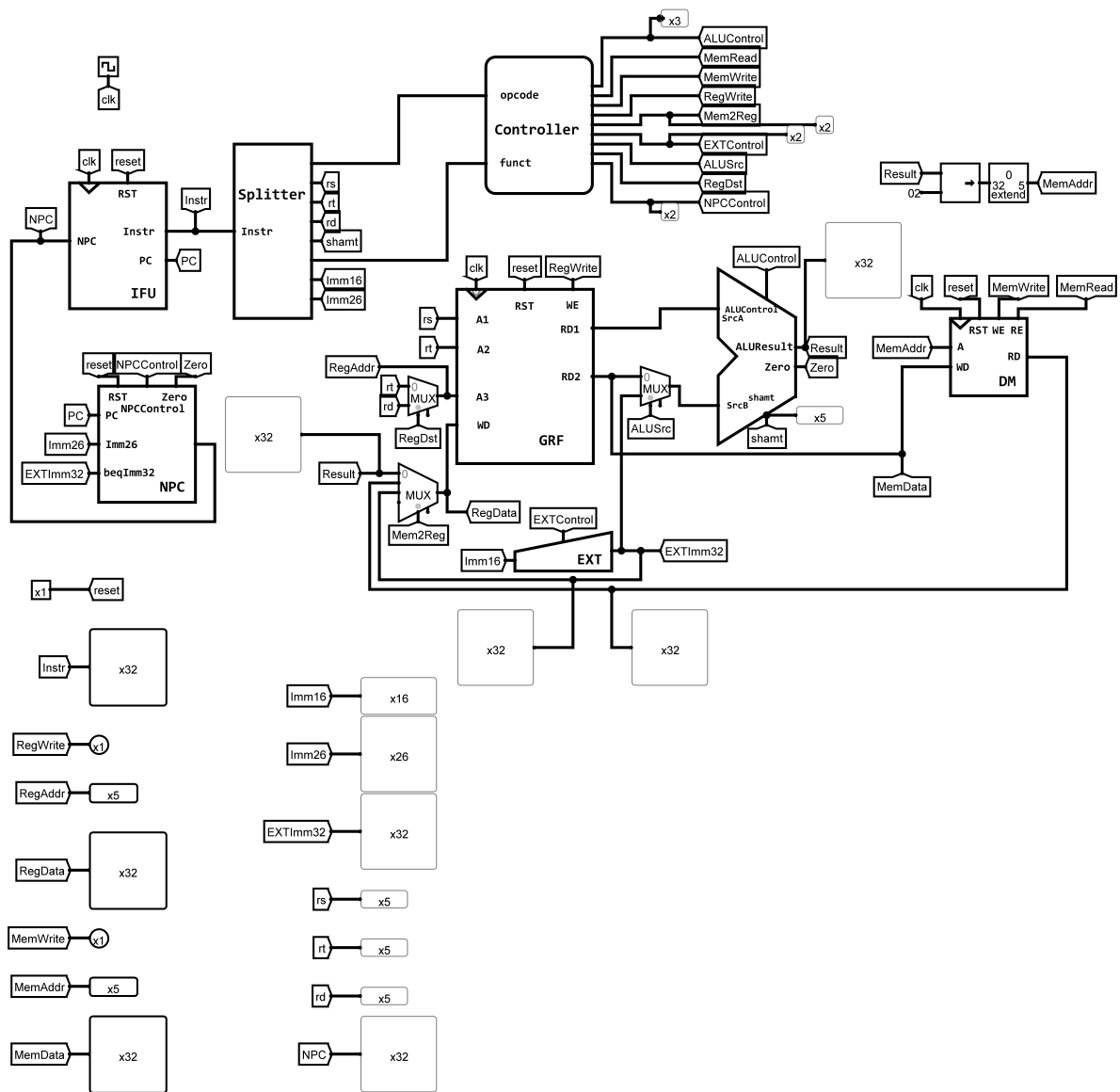
◦ 端口说明

序号	信号名	方向	描述
1	clk	I	内置时钟信号
2	reset	I	异步复位信号 1: 清零 0: 保持
3	MemWrite	I	写控制信号 1: 可写 0: 不可写
4	MemRead	I	读控制信号 1: 可读 0: 不可读
5	A[4:0]	I	5 位地址位选信号
6	WD[31:0]	I	写入内存的 32 位数据
7	RD[31:0]	O	从内存读出的 32 位数据

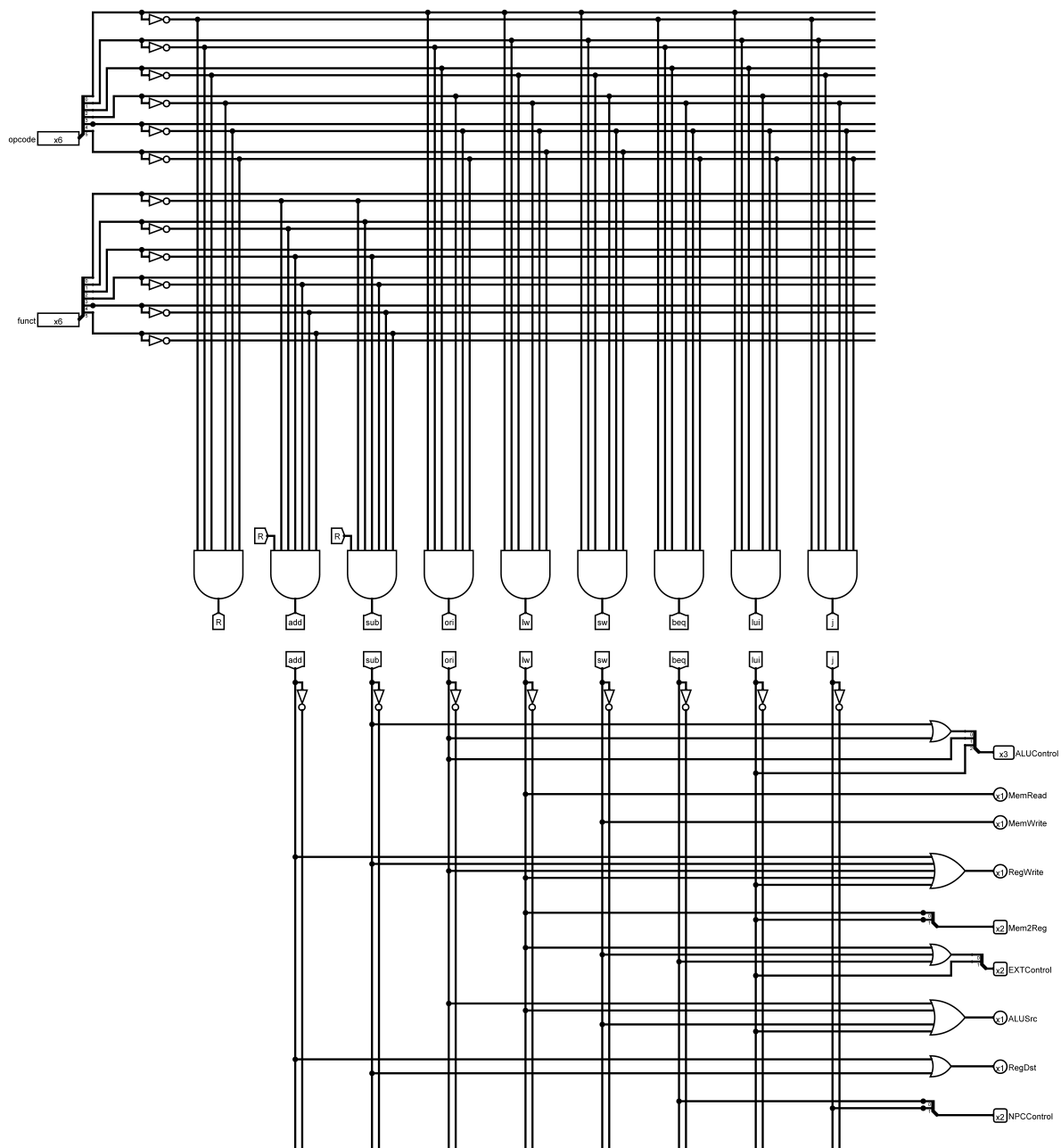
◦ 功能定义

序号	功能	描述
1	异步复位	reset 信号为 1 时，清空所有寄存器
2	读内存	MemRead 为 1 时，将 A 对应地址中的值读出到 RD
3	写内存	MemWrite 为 1 时，将 WD 的值写入到 A 对应的内存中

2.顶层电路外观



3.与或门阵列（控制器实现）



思考题

整体结构

思考题

上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

“上游”：PC 寄存器对应 FSM 中的状态存储模块，判断下一个存入 PC 寄存器的值的电路（通常封装为 NPC 模块）对应 FSM 中的状态转移电路，而从 ROM 中取出指令并传给 splitter 的电路对应 FSM 中的输出电路。

“下游”：从 splitter 输入指令的各段二进制码，由 Controller 生成控制信号的电路对应 FSM 中的状态转移电路，GRF 根据指令进行读写、改变其中存储的数据的电路对应 FSM 中的状态转移电路，而从 GRF 中输出下一个 PC 值的电路对应 FSM 中的输出电路。

模块规格

思考题

现在我们的模块中 IM 使用 ROM，DM 使用 RAM，GRF 使用 Register，这种做法合理吗？请给出分析，若有改进意见也请一并给出。

这种做法有一定的合理性，因为 IM 在我们的 CPU 对应的情况中只会存储一次，后续只需要读出即可；

DM 在我们的 CPU 对应的情况中会既读又写，所以使用 Logisim 中的 RAM 可以既读又写完成这些操作；

GRF 在我们的 CPU 对应的情况中会在时钟上升沿改变，且需要高速读写，对应多个寄存器的行为，所以使用多个 Register 实现较为合理。

这种做法也存在一定的局限性：

DM 在实际 CPU 使用过程中不可能只输入一次指令，这样会使得计算机载入新的程序时较为繁琐，限制了计算机的功能。在《数字设计与计算机体系结构》P229 的表述为：在处理指令存储器为 ROM 时，存在过度简化。在大多数实际处理器中，指令存储器必须是可写的，从而使操作系统可以载入一个新的程序到存储器中。

在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

设计了计算下一个 PC 的模块 NPC，用于计算 Next PC，这样 IFU 和 NPC 各司其职（IFU 只负责取指令，NPC 只负责计算 Next PC），满足“高内聚低耦合”的设计思想

设计思路为：NPC 会因指令不同而得到不同方式的计算结果，所以需要设计 NPCControl 控制信号进行选择。根据指令的不同，NPC 的计算方式可以分为三种：递增、branch 分支指令，j 跳转指令，根据指令集中对应的计算方式计算即可。

控制器设计

思考题

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？

因为 nop 空指令的机器码为 0x00000000，不会产生任何的高电平控制信号。nop 对应的周期中只会进行 $PC = PC + 4$ 的操作，不会进行其他任何操作，不会改变电路元器件存储的值，不需要单独加入真值表。

测试 CPU

思考题

上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以避免手工修改的麻烦。请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

阅读 Pre 的 [“MIPS 指令集及汇编语言”](#) 一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。

强度中等

指令种类覆盖全面，但单一指令各种行为及指令间协调一般。

如

```
ori $a0, $0, 123
ori $a1, $a0, 456
lui $a2, 123          # 符号位为 0
lui $a3, 0xffff       # 符号位为 1
```

此处未测试先进行ori再进行lui时低位的情况

再如

```
beq $a0, $a1, loop1    # 不相等
beq $a0, $a2, loop2    # 相等
loop1:sw $a0, 36($t0)
loop2:sw $a1, 40($t0)
```

检验不出beq的执行效果