P3 Document

设计草稿

NPC

NPCOp	NPC
00	PC+4
01	PC+4+sign_extend(offset||00)
10	PC31..28||instr_index||00
11	GPR[rs]

ALU

ALUOp	运算
00	加
01	减
10	或
11	与

EXT

EXTOp	扩展
00	零扩展
01	符号扩展
10	加载至高位

Controller

指令	RegWrite	RegDst	ALUSrc	MemWrite	WDSrc	NPCOp	ALUOp	EXTOp
R型 (000000+100xxx)	1	01	0	0	00	00	?11	00
ori(001101)	1	00	1	0	00	00	10	00
lw(100011)	1	00	1	0	01	00	00	01
sw(101011)	0	00	1	1	00	00	00	01
beq(000100)	0	00	0	0	00	01	01	00
lui(001111)	1	00	1	0	00	00	00	10
jal(000011)	1	10	0	0	10	10	00	00
jr(000000+001000)	0	00	0	0	00	11	00	00
注：R型指令的op段均为000000，ALUOp暂设为11								

RegDst

00: GPR[rt]

01: GPR[rd]

10: GPR[31]

ALUSrc

0: 来自于寄存器

1: 来自于立即数

WDSrc

00: 来自于ALU

01: 来自于DM

10: PC+4

R型指令可根据funct段得出各自的ALUOp信号。

R型	Funct	ALUOp
add	100000	00
sub	100010	01
or	100101	10
and	100100	11

测试方案

1. `ori` , `lui` , `add` , `sw` , `lw` , `beq`

```
.text
ori $a0, $0, 123
ori $a1, $a0, 456
lui $a2, 123
lui $a3, 0xffff
ori $a3, $a3, 0xffff
add $s0, $a0, $a2
add $s1, $a0, $a3
add $s2, $a3, $a3
ori $t0, $0, 0x0000
sw $a0, 0($t0)
sw $a1, 4($t0)
sw $a2, 8($t0)
sw $a3, 12($t0)
sw $s0, 16($t0)
sw $s1, 20($t0)
sw $s2, 24($t0)
lw $a0, 0($t0)
lw $a1, 12($t0)
sw $a0, 28($t0)
sw $a1, 32($t0)
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, loop1
beq $a0, $a2, loop2
loop1:sw $a0, 36($t0)
loop2:sw $a1, 40($t0)
```

机器码为

```
v2.0 raw
3404007b
```

```
348501c8
3c06007b
3c07ffff
34e7ffff
00868020
00878820
00e79020
34080000
ad040000
ad050004
ad060008
ad07000c
ad100010
ad110014
ad120018
8d040000
8d05000c
ad04001c
ad050020
34040001
34050002
34060001
10850001
10860001
ad040024
ad050028
```

2. `jal`, `jr`

```
.text
lui $a2, 0x7b
lui $a3, 0xffff
ori $a3, $a3, 0xffff
ori $a0, $zero, 1
jal jump
add $a2, $a2, $a0

jump:
sub $a2, $ra, $a1
jr  $ra
```

机器码为

```
v2.0 raw
3c06007b
3c07ffff
34e7ffff
34040001
0c000c0c
00c43020
03e53022
03e00008
```

思考题

1. 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。

> 状态存储：PC寄存器，GRF寄存器堆
> 状态转移：NPC，控制信号WDSrc控制的MUX相关电路

2. 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。

> 合理。ROM是只读存储器，适合用来储存指令，避免外部修改；RAM既可以读也可以写，满足了DM对 `sw` 和 `lw` 的要求；GRF是寄存器堆，需要较高的读写速度，因此适合用logisim中的Register实现。不足之处是RAM一次只能读写一个地址（4字节），因此无法直接实现 `lb` 和 `sb` 等指令。

3. 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NPC模块，给PC传递正确的地址。由于PC的下一个地址不只是PC+4，通过添加的控制信号NPCOp，实现b和j指令中对PC赋特殊值的功能，如 `beq` 需要 `PC+4+sign_extend(offset||00)` ， `jal` 需要 `PC31..28||instr_index||00` 。

4. 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？

> `nop` 指令信号为0，GRF写入地址为0号寄存器，不进行操作。不加入真值表，控制信号MemWrite=0，等效于空指令；NPCOp=0，PC=PC+4，执行下一条指令。

5. 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。

> 将MARS的存储位置设置为Text at Address 0，此时text的首地址为0，data在0x3000。设置一个片选信号，检测输入的地址的高4位，如果是0b0011，则说明是data中的数据，然后再导入到DM中。

6. 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。

没有空指令 `nop` ，本人在这里出错。 `sw` 和 `lw` 的offset没有负数情况， `beq` 没有往前跳转的情况。