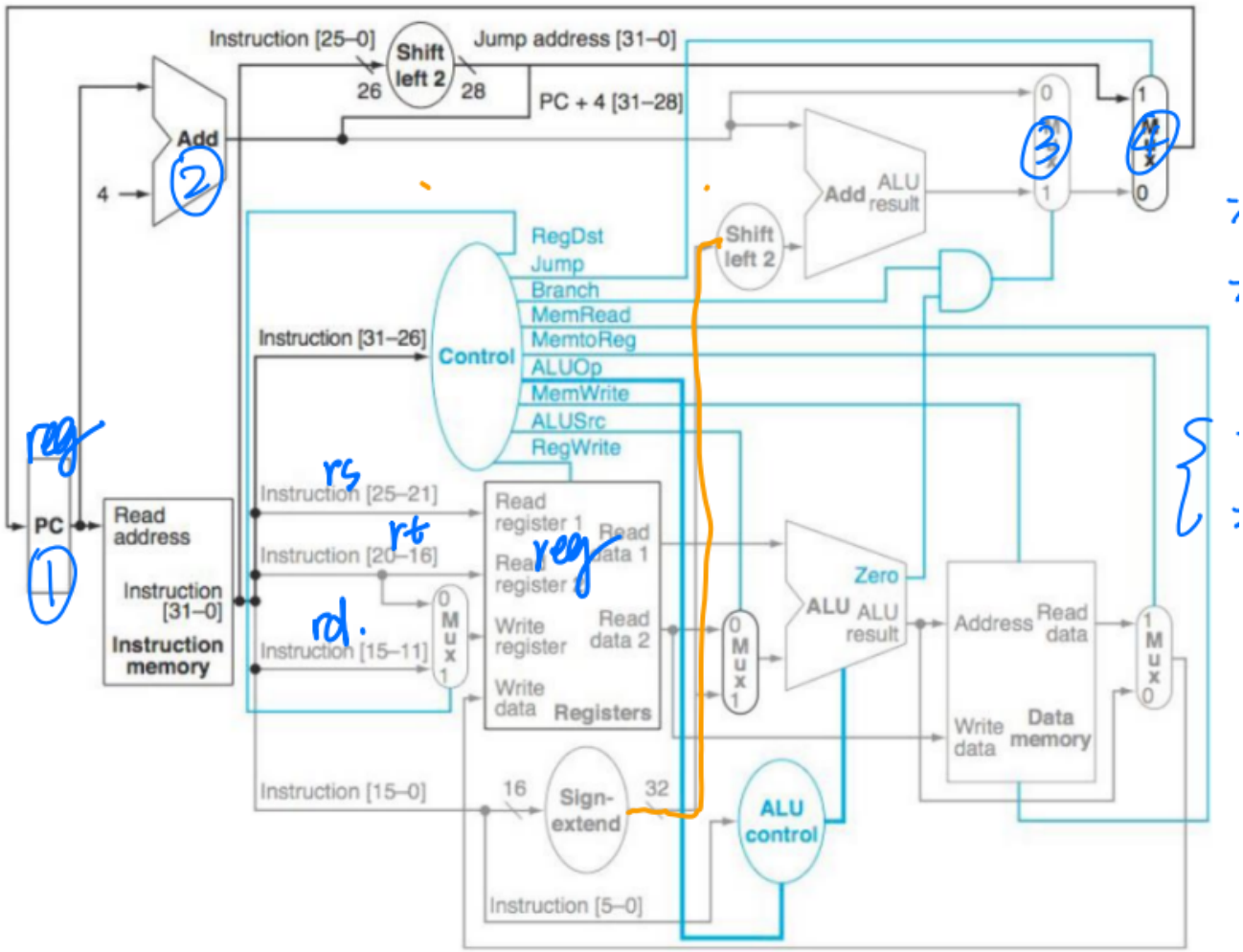
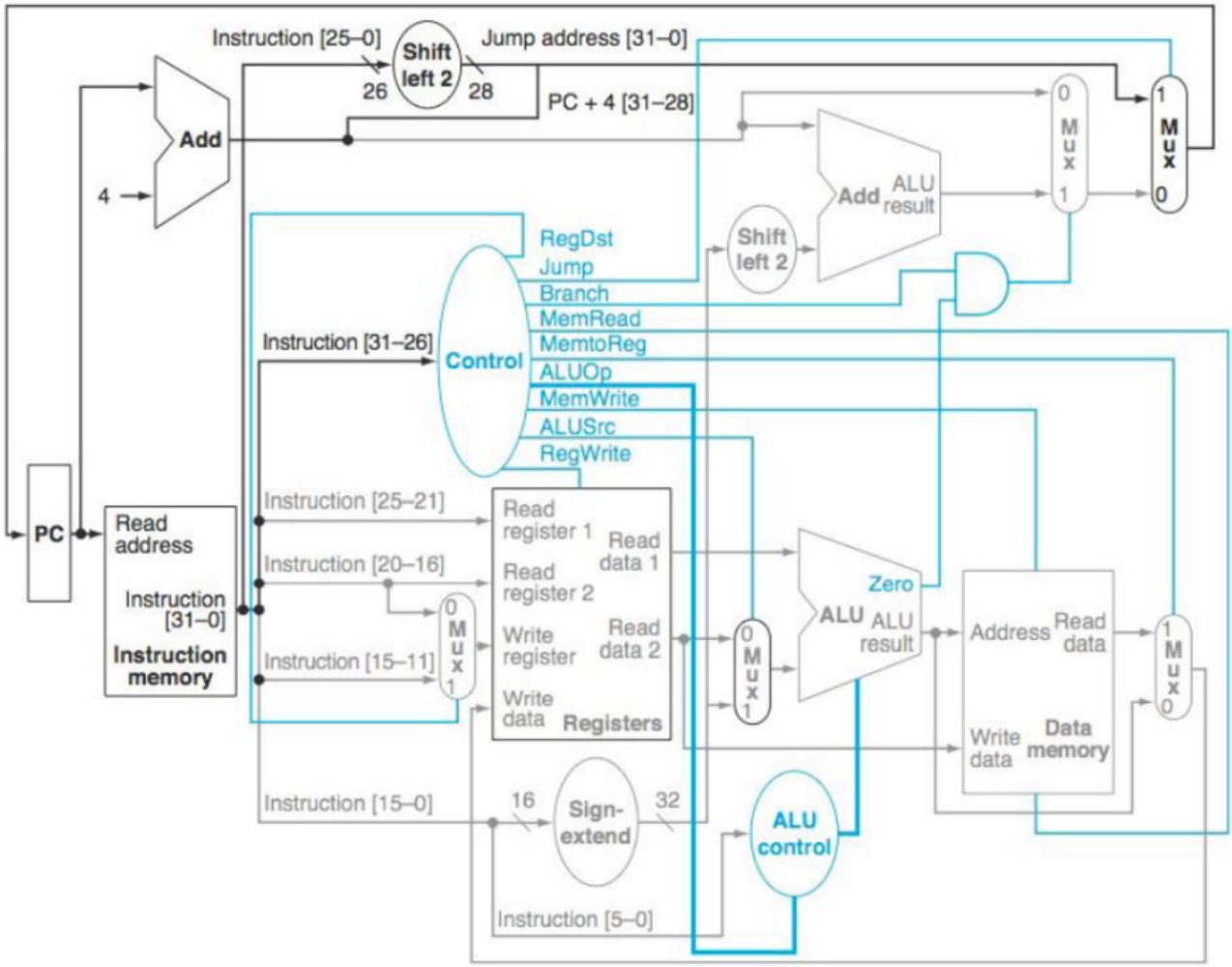
# 单周期CPU设计文档

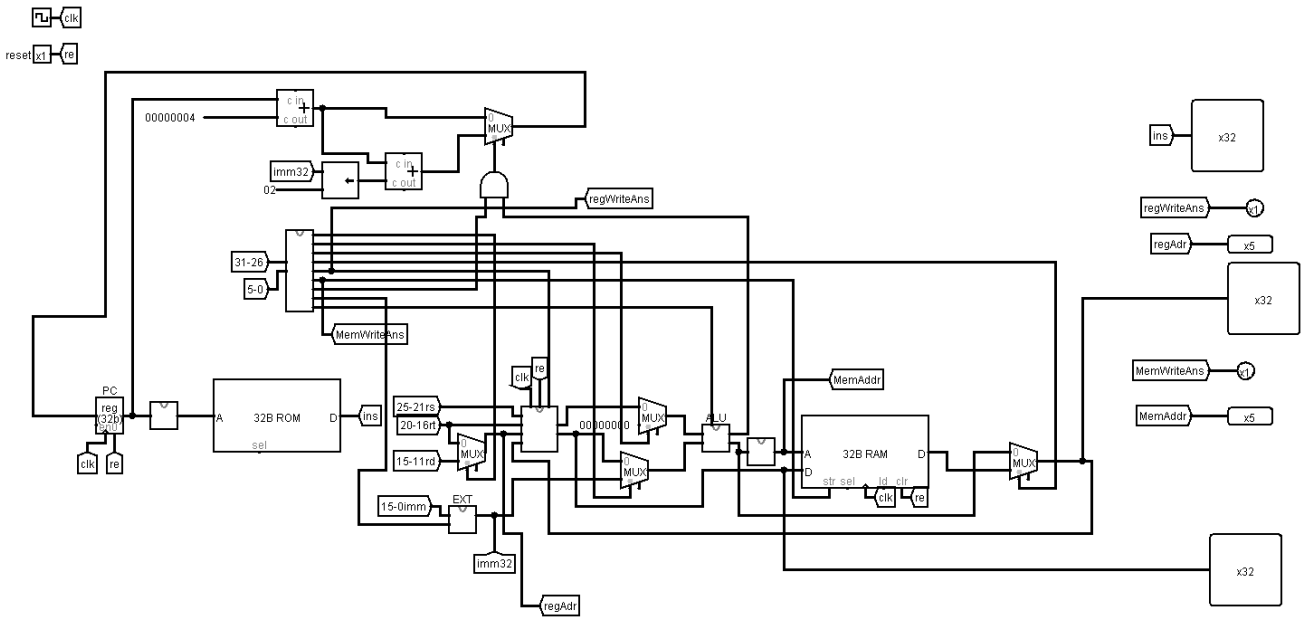
## 一、设计草稿

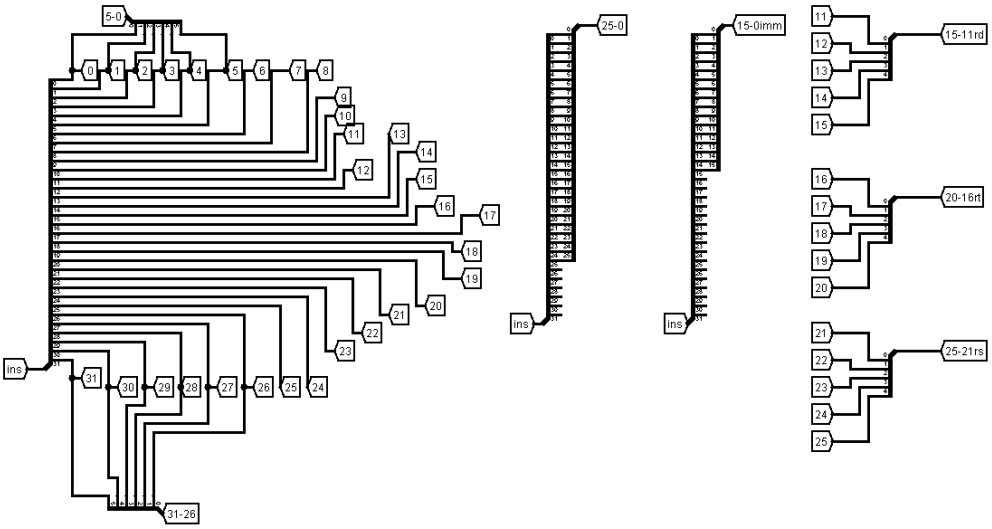
- 设计图




Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0 M u x 1
1 M u x 0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
ALU control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory

- 我设计出来的




- 控制信号

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| func（5-0） | 100000 | 100010 | | | | | |
| op（31-26） | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| 名称 | add | sub | ori | lw | sw | beq | lui |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| RegDst(有rd的指令需要为1）（mux的选择信号） | 1 | 1 | 0 | 0 | x | x | 0 |
| ALUSrc（选rt为0，选立即数为1）（mux的选择信号） | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| MemtoReg（选ALU结果为0，选内存DM读出结果为1）（mux的选择信号） | 0 | 0 | 0 | 1 | x | x | 0 |
| RegWrite（GRF的写入信号） | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| MemWrite（DM的写入信号） | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| branch（b类跳转信号） | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ExtOp<1:0>(控制拓展类型的信号）(00--0扩展，01--符号扩展，10--加载至高位) | x | x | 00 | 01 | 01 | x | 10 |
| ALUctr<2:0>(ALU运算类型信号）（000--加，001--减，010--or） | 000 | 001 | 010 | 000 | 000 | 001 | 000 |
| ALU2（其实意思应该是ALUSrc2）（选0为rs，选1为zero）（mux的选择信号） | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

# 二、测试方案

```
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
```

以上为mips导出的十六进制text

```
ori ````0, 123 ori ````a0, 456 lui ````a3, 0xffff # 符号位为 1 ori ````a3, 0xffff # ````s0, ````a2 # 正正 add ````a0, ````s2, ````a3 # 负负 ori ````0, 0x0000 sw ````t0) sw ````t0) sw ````t0) sw ````t0) sw ````t0) sw ````t0) sw ````t0) lw ````t0) lw ````t0) sw ````t0) sw ````t0) ori ````0, 1 ori ````0, 2 ori ````0, 1 beq ````a1, loop1 # 不相等 beq ````a2, loop2 # 相等 loop1:sw ````t0) loop2:sw ````t0)
```

以上为mips代码

对比寄存器和DM的值之后发现无误

# 三、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。

   发挥状态存储功能的有：GRF，PC

   发挥状态转移功能的有：其他

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通常用作硬盘，而RAM速度较快。GRF使用寄存器实现效率较高。

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。

   加入了一个取2-6位的模块方便取出地址

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？

   寄存器写入信号RegWrite和数据存储器读写信号MemWrite都是0，所以无法改变任何东西。

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。

   增加一个地址大于0x3000时减去0x3000的模块

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。

   对于lw指令没有测试目标寄存器是0时的状况，对于beq指令没有测试目标在此指令之前和就是此指令的情况