**一、CPU模块的总体设计**

1.CPU的主要性能指标和特性

* 地址/数据线宽度: 32bit
* 指令系统: RISC, 选取了Weihai MIPS中的16条
* 数据表示: 定点整数
* 寄存器组织:
  + PC
  + 32个通用寄存器
  + MDR、IR、A、B、ALUout
* 字节序: 大端次序
* 流水线: 不采用流水线
* 存储体系:
  + 存储字长: 32bit
  + 存储器划分: 按字编址，1K\*4B的ROM(0H开始)一块，1K\*4B的RAM（400H开始）一块
  + 结构: 普林斯顿结构
* I/O编址方式: 与主存统一编址(3FF0-3FF2)

2.CPU的构成及各部分的简要说明

* PC: 程序计数器，由于按字编址，每次+1
* MDR: 数据缓冲寄存器，暂存内存读出的数据
* IR: 指令寄存器，暂存内存读出的指令
* A,B: 暂存通用寄存器组的两个输出
* 通用寄存器组: 32个32位寄存器，可同时读2个，写1个
* ALUout: 暂存ALU的计算结果(不存标志位，仅存结果)
* ALU: 运算器，支持加、减、左移、右移、与、或、异或操作
* CU: 控制器，给出所有控制信号
* 内部数据通路: 包含大量多路选择器和信号线等硬件配置，完成指定位数的扩展操作等等
* 没有使用ALUControl，将op与funct全交CU处理

3.CPU的指令系统

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| xor | 000000 | rs | rt | rd(rs^rt) | 00000 | 100110 |
| sub | 000000 | rs | rt | rd(rs-rt) | 00000 | 100011 |
| add | 000000 | rs | rt | rd(rs+rt) | 00000 | 100000 |
| or | 000000 | rs | rt | rd(rs|rt) | 00000 | 100101 |
| addi | 001000 | rs | rt | imm16 | | |
| andi | 001100 | rs | rt | imm16 | | |
| xori | 001110 | rs | rt | imm16 | | |
| bne | 000101 | rs | rt | offset16 | | |
| beq | 000100 | rs | rt | offset16 | | |
| sw | 101011 | rs | rt | offset16 | | |
| lw | 100011 | rs | rt | offset16 | | |
| j | 000010 | target26 | | | | |
| jal | 000011 | target26 | | | | |
| jr | 000000 | rs | 000000000000000(15) | | | 001000 |
| sll | 000000 | 00000 | rt | rd | shamt5 | 000000 |
| srl | 000000 | 00000 | rt | rd | shamt5 | 000010 |

1. **R-type(xor, sub, add, or)**: 将rs和rt寄存器中的值按指令后六位操控ALU完成相应运算后存储结果到rd寄存器。（发生溢出不中断）

2. **立即数(addi, andi, xori)**: 将16位立即数符号扩展到32位后与rs寄存器中的值按指令前六位操作码操控ALU完成相应运算后存储结果到rt寄存器。（发生溢出不中断）

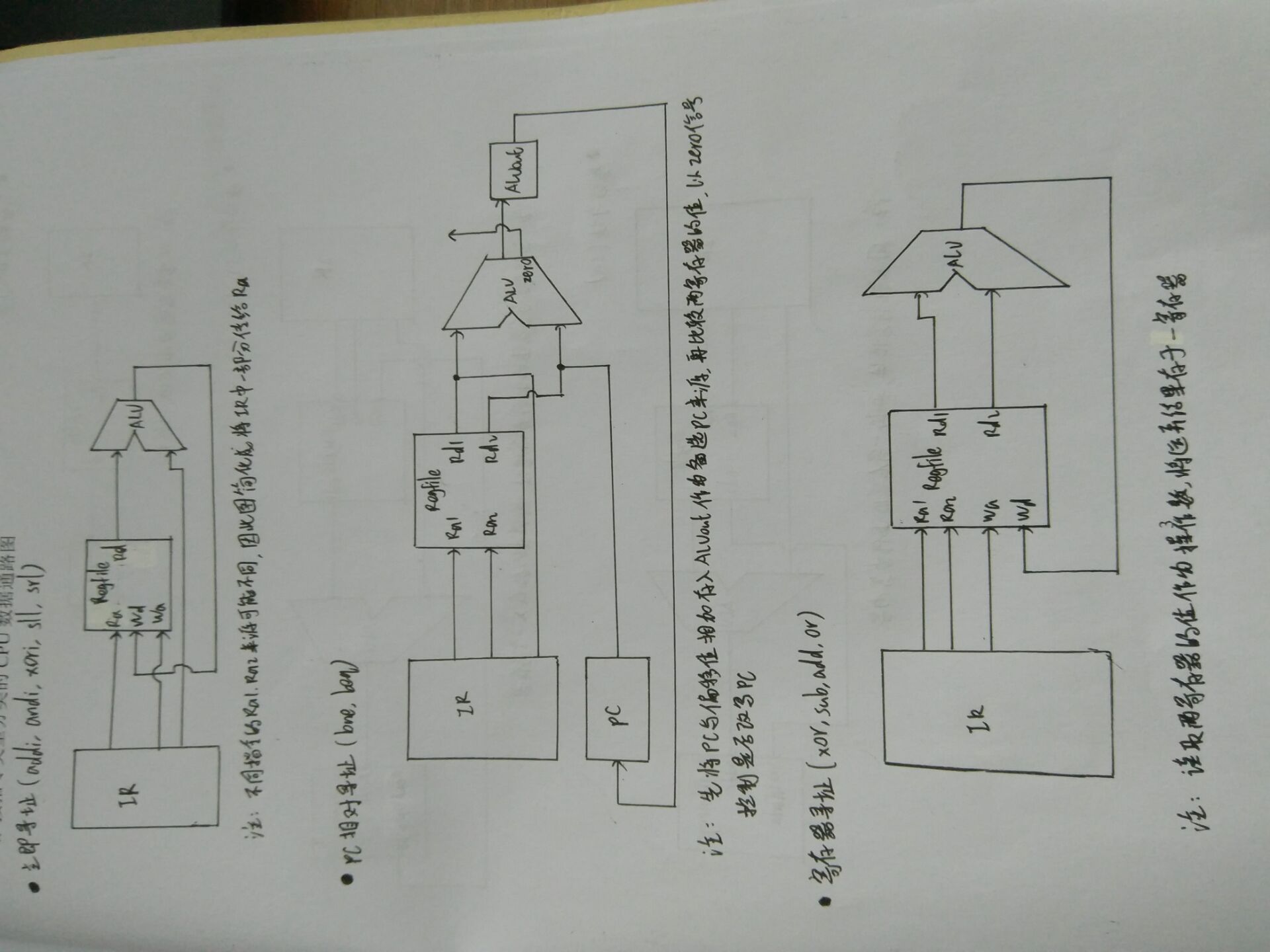
3. **Branch(bne, beq)**: 根据rs和rt寄存器中值的相等与否跳转到地址——PC当前值+16位offset符号扩展到32位。

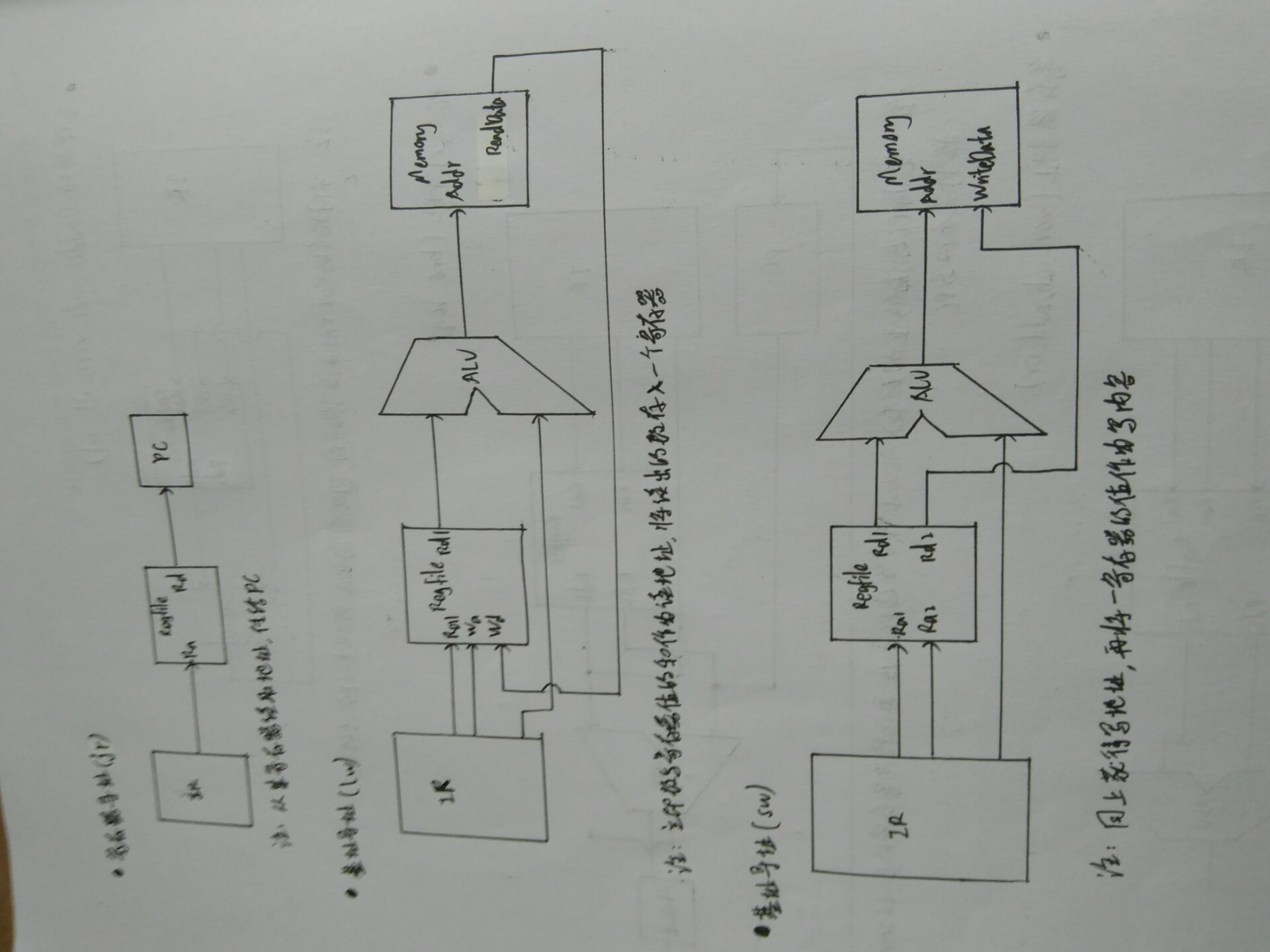
4. **内存读写(lw, sw)**: Lw向RAM地址——rs寄存器值+16位offset符号扩展到32位，写入rt寄存器的值（或sw:将该地址值读出并写入到rt寄存器）。

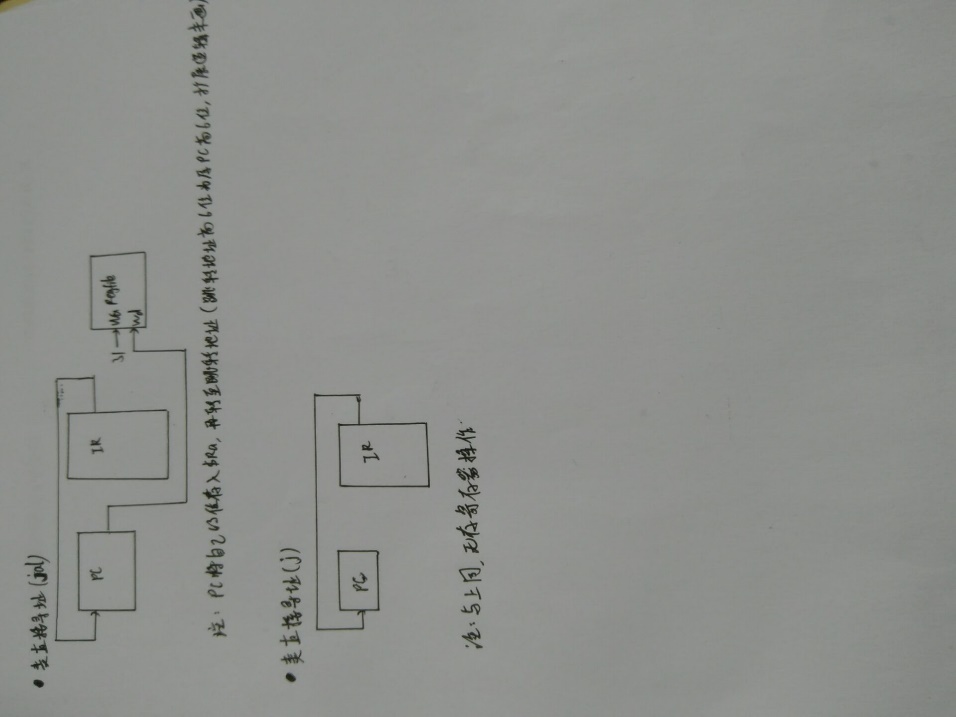
5. **jump(j, jal, jr)**: j, jal跳转到地址——PC的高6位接上26位target, jr跳转到地址——rs寄存器的值；Jal跳转前还将当前PC值存入了31号通用寄存器。

6. **移位(sll, srl)**: 将rt寄存器的值按指令后六位操控ALU逻辑移位无符号数shamt位，并将ALU运算结果存入rd寄存器。

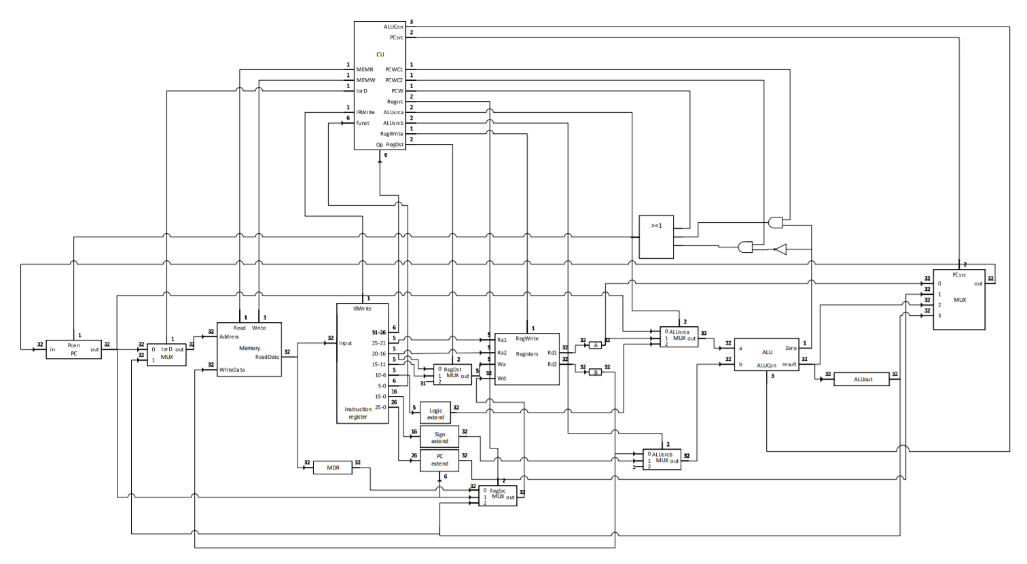
4.按指令类型分类的CPU数据通路图







5.总体的CPU数据通路图



6.CPU主要模块的控制信号分析

|  |  |  |
| --- | --- | --- |
| ALUCon | 000 | 加法 |
|  | 001 | 左移 |
|  | 010 | 右移 |
|  | 011 | 异或 |
|  | 100 | 与 |
|  | 101 | 或 |
|  | 110 | 减 |
| PCsrc | 00 | pc来自A寄存器（对应jr指令） |
|  | 01 | pc来自pc-extend（对应无条件转移指令j和jar） |
|  | 10 | pc直接来自运算器结果（对应pc = pc + 1） |
|  | 11 | pc来自ALUOut寄存器（对应条件转移指令beq和bne） |
| PCWC1 | 0 | / |
|  | 1 | 对应相等则转移（beq） |
| PCWC2 | 0 | / |
|  | 1 | 对应不相等则转移（bne） |
| PCW | 0 | / |
|  | 1 | 对应无条件转移(j jal jr pc+1) |
| Regsrc | 00 | Wd来自MDR（对应读内存指令lw） |
|  | 01 | Wd来自pc（对应jal指令） |
|  | 10 | Wd来自ALUOut（对应所有计算性结果） |
| ALUsrca | 00 | A操作数来自pc （对应beq bne pc + 1 ） |
|  | 01 | A操作数来自A寄存器 (branch也用) |
|  | 10 | A操作数来自IR[10:6]（对应移位指令） |
| ALUsrcb | 00 | B操作数来自B寄存器（branch也用） |
|  | 01 | B操作数来自sign-extend（对应bne beq lw sw \*\*\*i） |
|  | 10 | B操作数是1（pc = pc + 1） |
| RegWrite | 0 | / |
|  | 1 | Regfile可以写 |
| RegDst | 00 | Wa来自IR[20:16] |
|  | 01 | Wa来自IR[15:11] (对应and or sll srl) |
|  | 10 | Wa为31（对应jal存pc当前值到$ra） |
| MEMW | 0 | / |
|  | 1 | Memory可以写 |
| MEMR | 0 | / |
|  | 1 | Memory可以读 |
| IorD | 0 | 访存地址来自PC（对应取指周期） |
|  | 1 | 访存地址来自ALUOut（对应sw） |
| IRWrite | 0 | / |
|  | 1 | IR可以写（对应取指周期） |

7.多周期操作设计(每个周期完成功能及控制信号电平分析)

（为了访存稳定，存储器工作频率为100MHz，CPU工作频率为50MHz，防止一个节拍后存储器读数还未稳定）

**公有周期**：

**节拍1. 取指周期**

信号电平：ALUCon=000 PCsrc=10 PCWC1=0 PCWC2=0 PCW=1 ALUsrca=00 ALUsrcb=10 RegWrite=0 MEMW=0 MEMR=1 IORD=0 IRWrite=1

完成功能：

①根据当前PC值从存储器取出指令到IR寄存器

②完成pc = pc + 1

**节拍2. 指令解析和寄存器读数周期**

信号电平：ALUCon=000 PCWC1=0 PCWC2=0 PCW=0 ALUsrca=00 ALUsrcb=01 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①CU解析指令，对不同指令指定不同的“下一个”状态(详见CU代码)

②在ALU中完成（pc + 符号扩展的IR[15: 0]），存入ALUOut寄存器，以备后续bne，beq指令使用，若实际上本条指令不是bne或beq，这个计算结果是没有用处的，但是此处计算相比于闲置ALU来说也没有任何坏处(反正ALU是组合逻辑电路)。

③读出IR[25:21]和IR[20:16]号通用寄存器的值，分别写入A,B寄存器。

**非公有周期：**

**对于基址寻址之lw:**

**节拍3. 访存地址计算周期**

信号电平：ALUCon=000 PCWC1=0 PCWC2=0 PCW=0 ALUsrca=01 ALUsrcb=01 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①A寄存器中的值作为基址与符号扩展的IR[15: 0]在ALU中完成相加，存入ALUOut寄存器。

**节拍4. 读存周期**

信号电平：PCWC1=0 PCWC2=0 PCW=0 RegWrite=0 MEMW=0 MEMR=1 IRWrite=0 IorD=1

完成功能：

①根据ALUOut中的地址进行内存读操作，读出的数据存入MDR。

**节拍5. 读内存完成周期**

信号电平：PCWC1=0 PCWC2=0 PCW=0 RegWrite=1 MEMW=0 MEMR=0 IRWrite=0 RegDst=00 Regsrc=00

完成功能：

①将MDR中数据写入IR[20:16]号寄存器。

**对于基址寻址之sw:**

**节拍3. （同lw）**

**节拍4. 写存周期**

信号电平：PCWC1=0 PCWC2=0 PCW=0 RegWrite=0 MEMW=1 MEMR=0 IRWrite=0 IorD=1

完成功能：

①根据ALUOut中的地址将B寄存器中的值写入内存。

**对于寄存器寻址之R-type类：**

**节拍3. R-type执行周期**

信号电平：ALUCon=000/101/011/110 PCWC1=0 PCWC2=0 PCW=0 ALUsrca=01 ALUsrcb=00 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①根据不同ALUCon信号在ALU中对寄存器A和B的值做操作后，存入ALUOut寄存器。

**节拍4. R-type完成周期**

信号电平：PCWC1=0 PCWC2=0 PCW=0 RegDst=01 Regsrc=10 RegWrite=1 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①将ALUOut中的值写入IR[15:11]号寄存器。

**对于立即寻址之移位类：**

**节拍3. 移位执行周期**

信号电平：ALUCon=001/010 PCWC1=0 PCWC2=0 PCW=0 ALUsrca=10 ALUsrcb=00 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①根据ALUCon将B寄存器中的值左（或右）移逻辑扩展的IR[10:6]位，存入ALUOut寄存器。

**节拍4. （与R-type相同）**

**对于立即寻址之立即数类：**

**节拍3. 立即数类执行周期**

信号电平：ALUCon=000/011/100 PCWC1=0 PCWC2=0 PCW=0 ALUsrca=01 ALUsrcb=01 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①根据不同ALUCon信号在ALU中对寄存器A的值和符号扩展的IR[15:0]做操作后，存入ALUOut寄存器。

**节拍4. 立即数类完成周期**

信号电平：PCWC1=0 PCWC2=0 PCW=0 RegDst=00 Regsrc=10 RegWrite=1 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①将ALUOut中的值写入IR[20:16]号寄存器。

**对于PC相对寻址类：**

**节拍3. 分支类完成周期**

信号电平：ALUCon=110 PCWC1=0/1 PCWC2=~PCWC1 PCW=0 PCsrc=11 ALUsrca=01 ALUsrcb=00 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①将A，B寄存器的值在ALU中做减法。

②根据ALU的Zero标志输出用ALUOut寄存器的“当前值”覆盖PC。

**对于类直接寻址之jal：**

**节拍3. 程序调用完成周期**

信号电平：PCW=1 PCsrc=01 RegWrite=1 MEMW=0 MEMR=0 IRWrite=0 RegDst=10 Regsrc=01

完成功能：

①用IR[25:0]覆盖PC。

②将PC“当前值”存入31号通用寄存器。

**对于类直接寻址之j：**

**节拍3. 无条件转移完成周期**

信号电平：PCW=1 PCsrc=01 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

完成功能：

①用IR[25:0]覆盖PC。

**对于寄存器寻址之jr：**

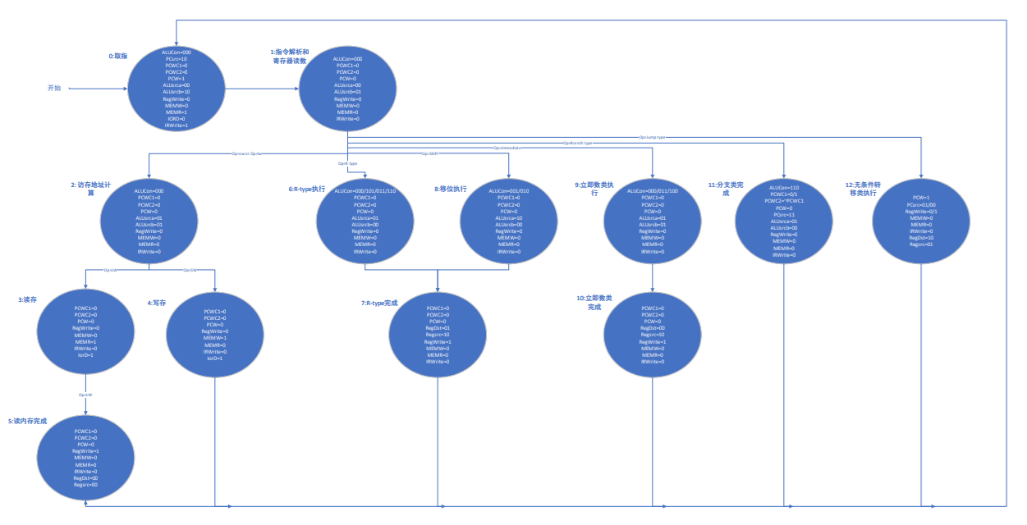
**节拍3. 程序返回完成周期**

信号电平：PCW=1 PCsrc=00 RegWrite=0 MEMW=0 MEMR=0 IRWrite=0

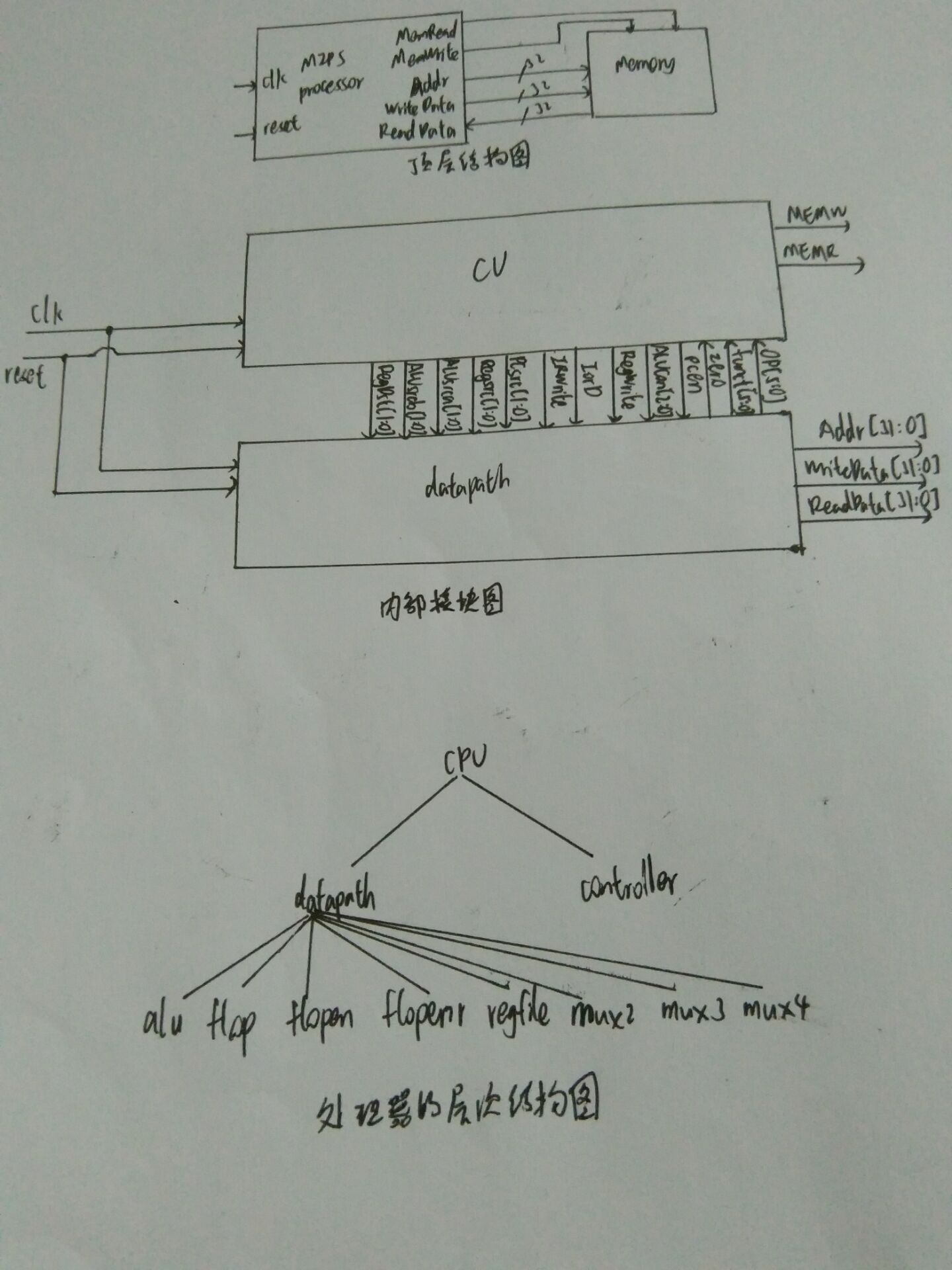
完成功能：

①用A寄存器的“当前值”覆盖PC。

8.控制单元的状态转换图

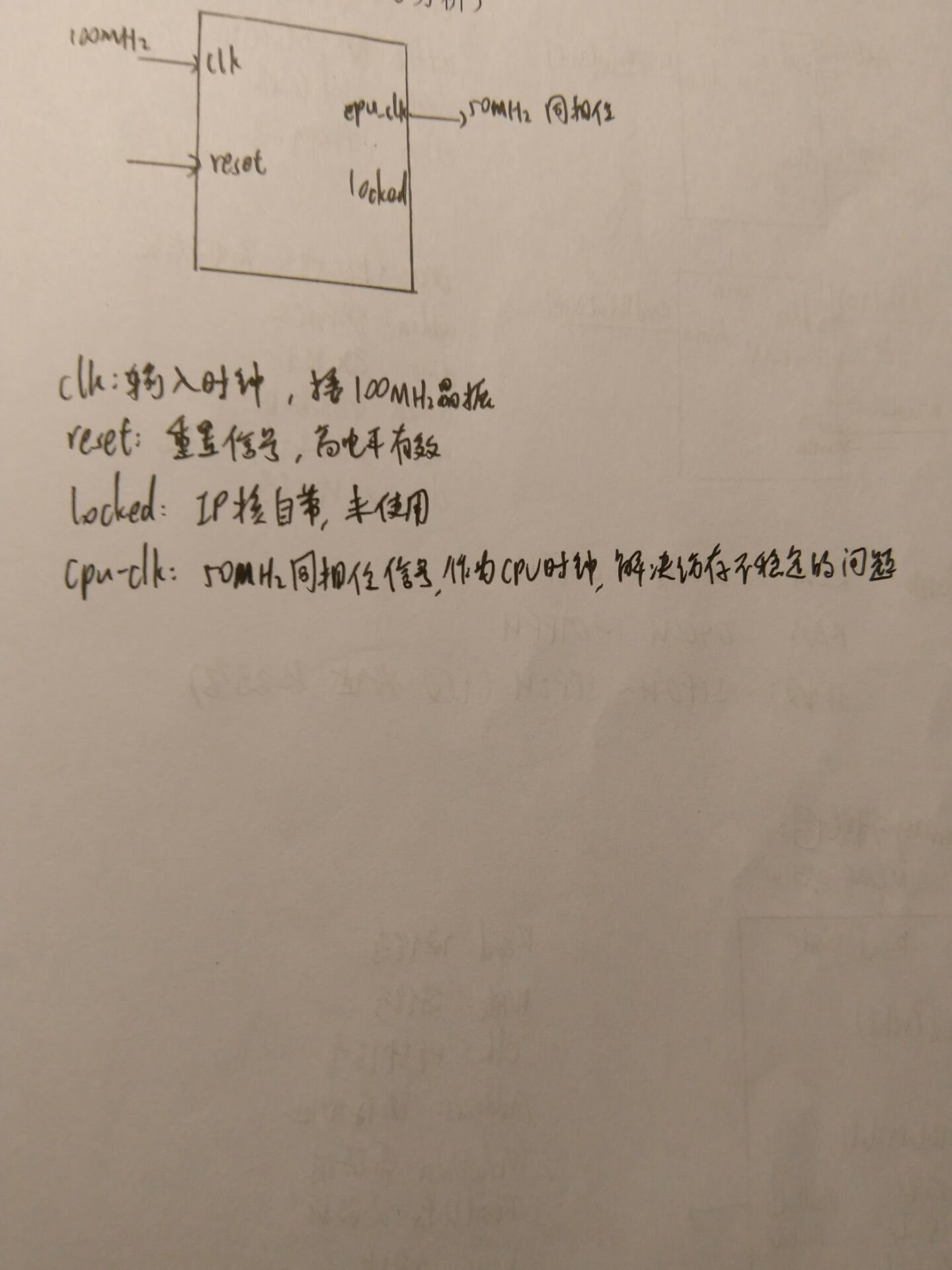


9.CPU的Verilog描述分析(模块构成及层次图)

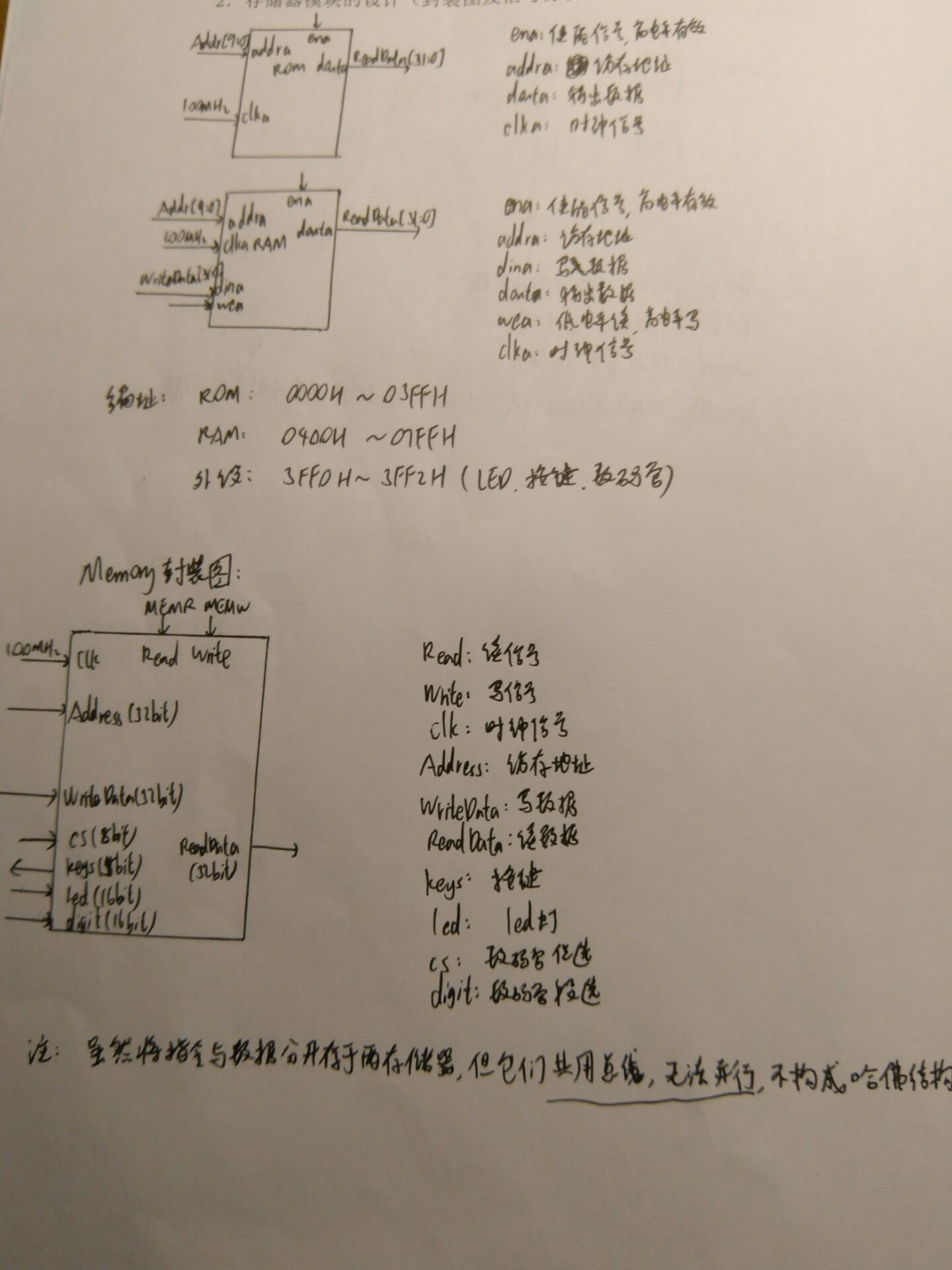


**二、外围模块的设计**

1.时钟模块的设计(封装图及信号分析)

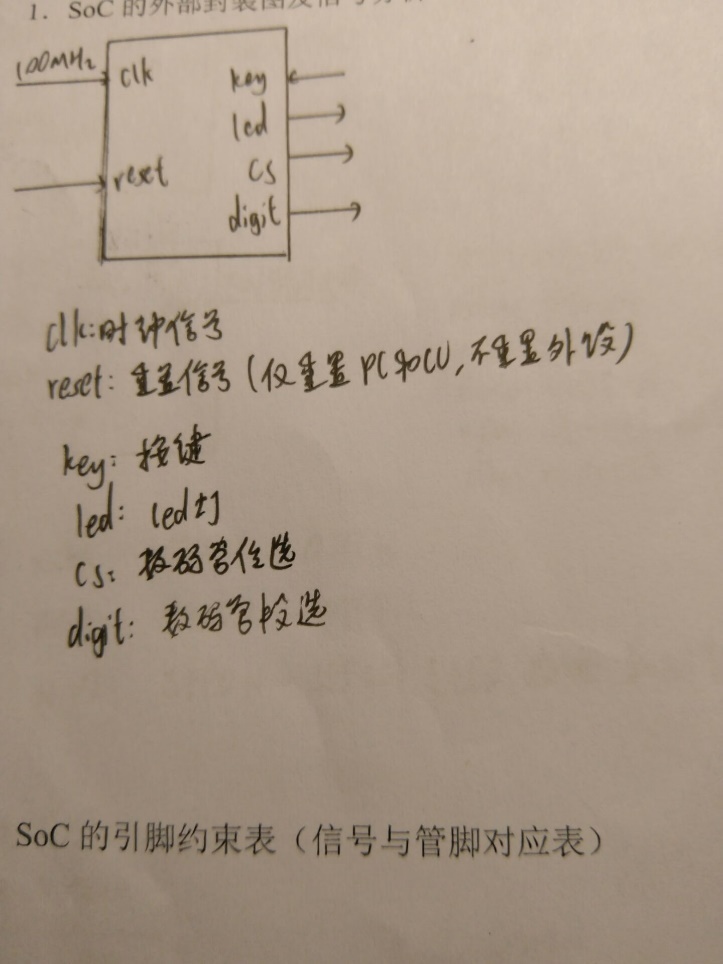


2.存储器模块的设计(封装图及信号分析)



**三、SOC的设计**

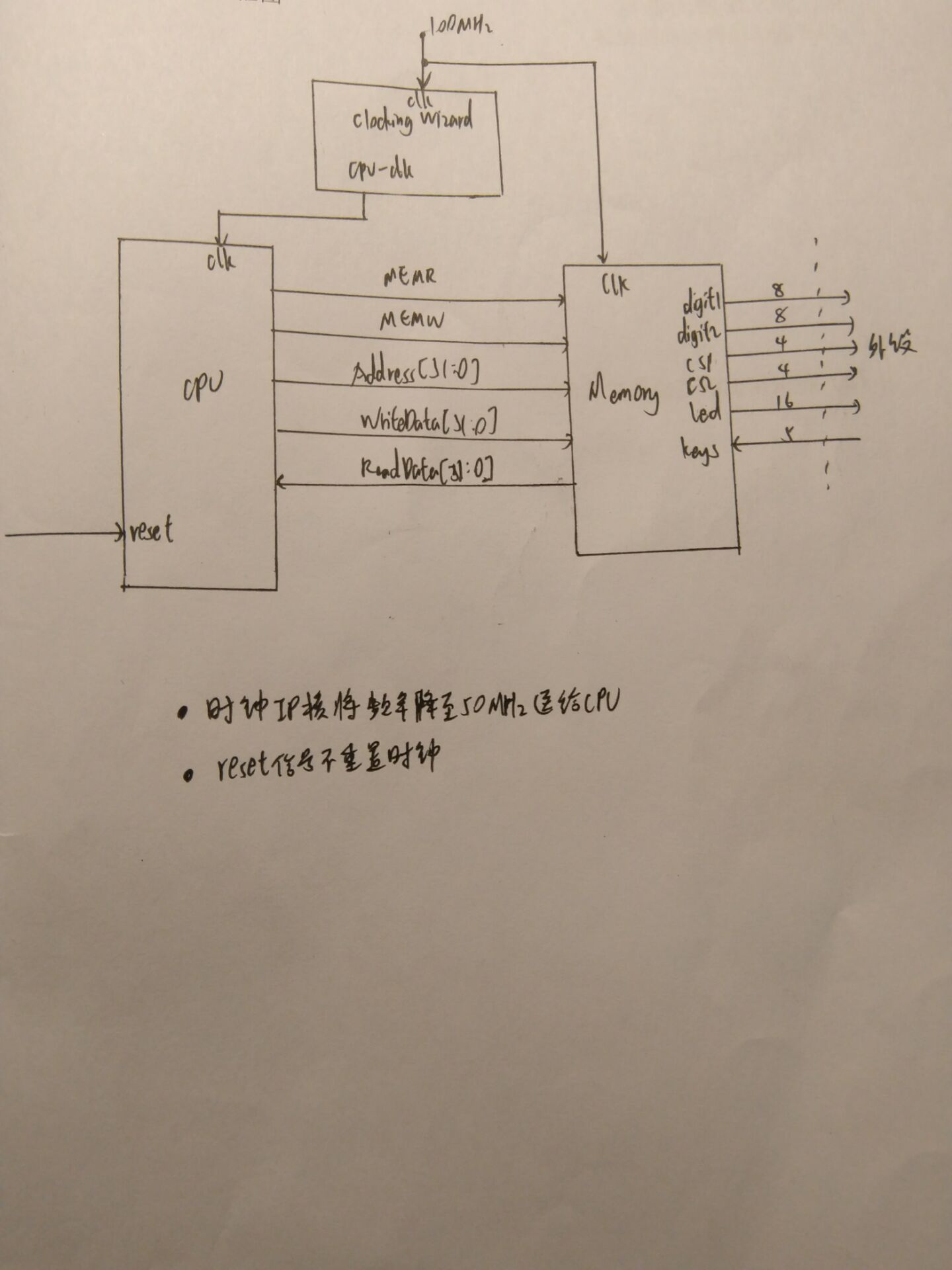
1.SoC的外部封装图及信号分析(输入输出信号)



2.SoC的引脚约束表(信号与管脚对应表)

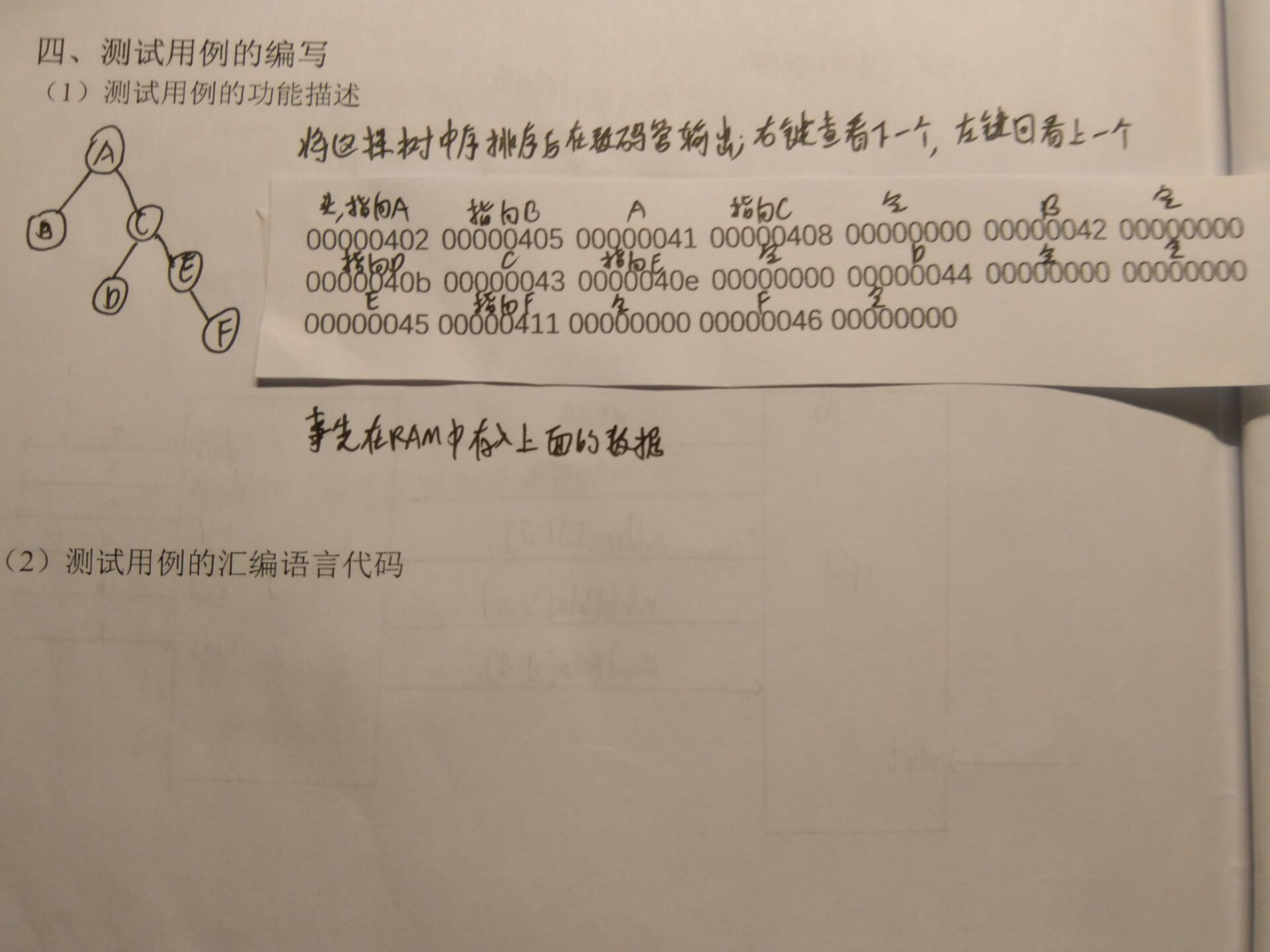
|  |  |  |
| --- | --- | --- |
| Name | Direction | Package Pin |
| Ports (47) | | |
| clk | IN | P17 |
| cs1[3] | OUT | G2 |
| cs1[2] | OUT | C2 |
| cs1[1] | OUT | C1 |
| cs1[0] | OUT | H1 |
| cs2[3] | OUT | G1 |
| cs2[2] | OUT | F1 |
| cs2[1] | OUT | E1 |
| cs2[0] | OUT | G6 |
| digit1[7] | OUT | B4 |
| digit1[6] | OUT | A4 |
| digit1[5] | OUT | A3 |
| digit1[4] | OUT | B1 |
| digit1[3] | OUT | A1 |
| digit1[2] | OUT | B3 |
| digit1[1] | OUT | B2 |
| digit1[0] | OUT | D5 |
| digit2[7] | OUT | D4 |
| digit2[6] | OUT | E3 |
| digit2[5] | OUT | D3 |
| digit2[4] | OUT | F4 |
| digit2[3] | OUT | F3 |
| digit2[2] | OUT | E2 |
| digit2[1] | OUT | D2 |
| digit2[0] | OUT | H2 |
| key[4] | IN | R11 |
| key[3] | IN | V1 |
| key[2] | IN | R17 |
| key[1] | IN | U4 |
| key[0] | IN | R15 |
| led[15] | OUT | K3 |
| led[14] | OUT | M1 |
| led[13] | OUT | L1 |
| led[12] | OUT | K6 |
| led[11] | OUT | J5 |
| led[10] | OUT | H5 |
| led[9] | OUT | H6 |
| led[8] | OUT | K1 |
| led[7] | OUT | K2 |
| led[6] | OUT | J2 |
| led[5] | OUT | J3 |
| led[4] | OUT | H4 |
| led[3] | OUT | J4 |
| led[2] | OUT | G3 |
| led[1] | OUT | G4 |
| led[0] | OUT | F6 |
| reset | IN | P15 |

3.SoC的结构框图



**三、测试用例的编写**

(1)测试用例的功能描述



(2)测试用例的汇编语言代码

addi $sp,$zero,1500 #sp初始指向500

addi $t1,$zero,1600 #存放结果的头位置

wait:

lw $t8, 0x3ff1($zero) #取按键内容

andi $t4, $t8, 0xffff #是否有按键

beq $t4, $zero, wait #若无则继续查询等待

jal delay #延时一段时间

lw $t9, 0x3ff1($zero) #取按键内容

beq $t8, $t9, wait #与25ms前一致则重新等待按键(下降沿触发)

#根据按键进入不同程序

andi $t7, $t4, 0x0001

bne $t7, $zero, center

andi $t7, $t4, 0x0008

bne $t7, $zero, left

andi $t7, $t4, 0x0010

bne $t7, $zero, right

j wait

center:

andi $a0, $a0, 0

addi $a0,$zero,1026 #子程序参数

jal travel #进行中序遍历函数

#数码管显示"in"

andi $t6, $t6, 0

addi $t6, $t6, 0x0160

sll $t6, $t6, 16

addi $t6, $t6, 0x08ec

sw $t6, 0x3ff2($zero)

andi $t2, $t2, 0

addi $t2,$t1,-1600 #存放结果长度

andi $t1, $t1, 0

addi $t1,$zero,1599 #存放结果的头位置

j wait

left:

#数码管清零

andi $t6, $t6, 0

sw $t6, 0x3ff2($zero)

#读取指针-1位置的字母

addi $t1,$t1,-1

lw $t0, 0($t1)

j seg

seg:

if\_a:

andi $t5, $t5, 0

addi $t5, $t5, 0x0041

bne $t0, $t5, if\_b

andi $t6, $t6, 0

addi $t6, $t6, 0x08ee

sw $t6, 0x3ff2($zero)

j wait

if\_b:

andi $t5, $t5, 0

addi $t5, $t5, 0x0042

bne $t0, $t5, if\_c

andi $t6, $t6, 0

addi $t6, $t6, 0x083e

sw $t6, 0x3ff2($zero)

j wait

if\_c:

andi $t5, $t5, 0

addi $t5, $t5, 0x0043

bne $t0, $t5, if\_d

andi $t6, $t6, 0

addi $t6, $t6, 0x089c

sw $t6, 0x3ff2($zero)

j wait

if\_d:

andi $t5, $t5, 0

addi $t5, $t5, 0x0044

bne $t0, $t5, if\_e

andi $t6, $t6, 0

addi $t6, $t6, 0x087a

sw $t6, 0x3ff2($zero)

j wait

if\_e:

andi $t5, $t5, 0

addi $t5, $t5, 0x0045

bne $t0, $t5, if\_f

andi $t6, $t6, 0

addi $t6, $t6, 0x089e

sw $t6, 0x3ff2($zero)

j wait

if\_f:

andi $t5, $t5, 0

addi $t5, $t5, 0x0046

bne $t0, $t5, if\_g

andi $t6, $t6, 0

addi $t6, $t6, 0x088e

sw $t6, 0x3ff2($zero)

j wait

if\_g:

andi $t5, $t5, 0

addi $t5, $t5, 0x0047

bne $t0, $t5, other

andi $t6, $t6, 0

addi $t6, $t6, 0x08f6

sw $t6, 0x3ff2($zero)

j wait

other:

j wait

right:

#数码管清零

andi $t6, $t6, 0

sw $t6, 0x3ff2($zero)

#读取指针+1位置的字母

addi $t1,$t1,1

lw $t0, 0($t1)

j seg

travel:

addi $sp, $sp, -2 #修/改堆栈指针，准备入栈

sw $ra, 1($sp) #返回地址入栈

sw $a0, 0($sp) #参数入栈

beq $a0, $zero, pop #节点为空，出栈；不为空，其左节点继续执行函数

lw $t0, -1($a0) #将$a0存储的字母的左子树的地址存入$t0

addi $a0, $t0, 0 #参数存入$a0

jal travel #继续执行travel

sll $zero, $zero, 0

lw $ra, 1($sp) #恢复本轮调用的返回地址至$ra

lw $a0, 0($sp) #将栈指针指向的字母指针存入a0

lw $t0, 0($a0) #将a0指向的字母值存入t0

sw $t0, 0($t1) #把字母存入主存某位置

addi $t1, $t1, 1 #存字母的主存地址位置+1

lw $t0, 1($a0) #将$a0存储的字母的右子树的地址存入$t0

addi $a0, $t0, 0 #参数存入$a0

jal travel #继续执行travel

sll $zero, $zero, 0

lw $ra, 1($sp) #恢复本轮调用的返回地址至$ra

pop:

addi $sp, $sp, 2

jr $ra

delay:

andi $t9, $t9, 0

addi $t9, $t9, 1 #延迟25ms

sll $t9, $t9, 18

minus:

addi $t9, $t9, -1

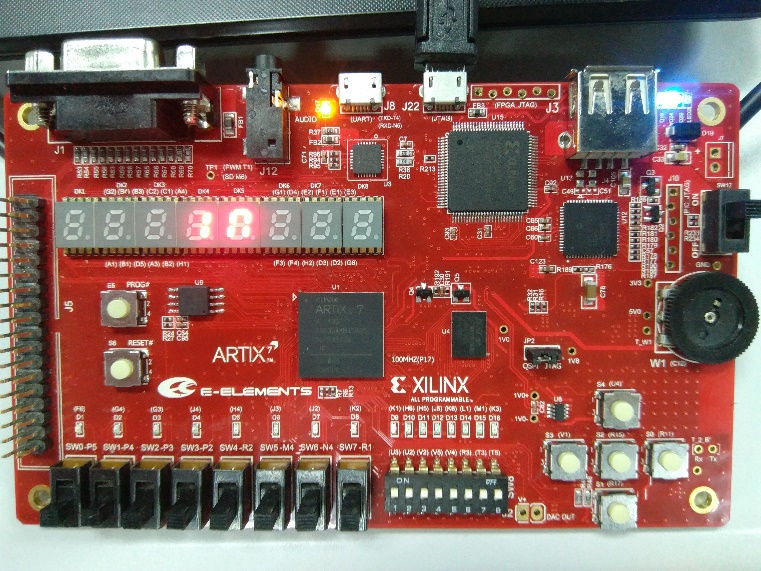
bne $t9, $zero, minus

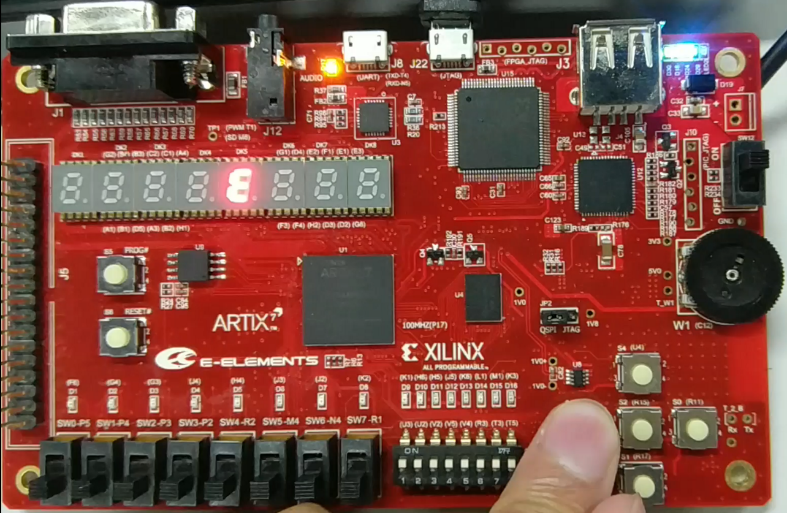
jr $ra

(3)测试用例的机器语言代码

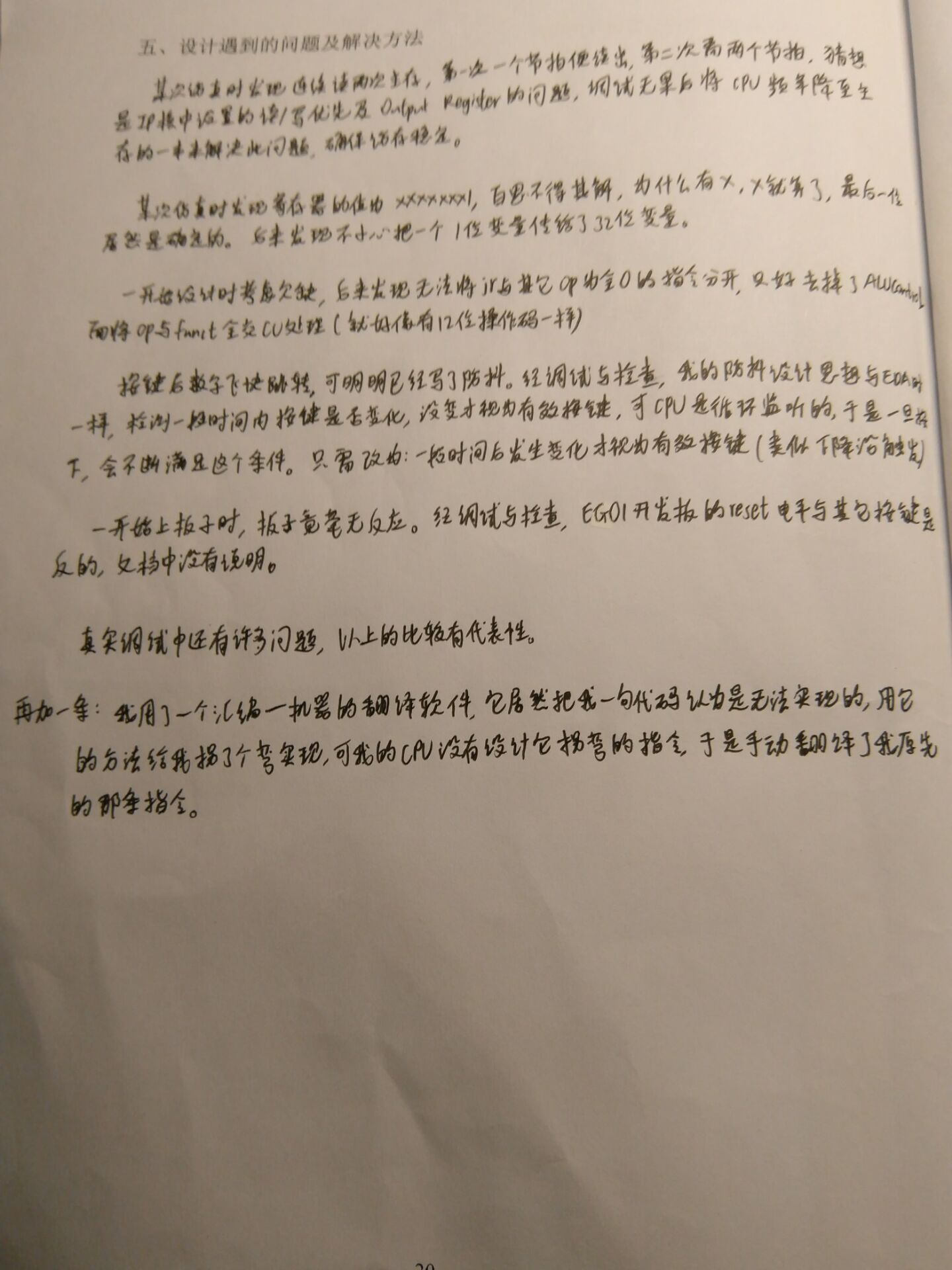
1. 201d05dc
2. 20090640
3. 8c183ff1
4. 330cffff
5. 1180fffd
6. 0c10006c
7. 8c193ff1
8. 1319fffa
9. 318f0001
10. 15e00005
11. 318f0008
12. 15e00010
13. 318f0010
14. 15e00045
15. 08100002
16. 30840000
17. 20040402
18. 0c100058
19. 31ce0000
20. 21ce0160
21. 000e7400
22. 21ce08ec
23. ac0e3ff2
24. 314a0000
25. 212af9c0
26. 31290000
27. 2009063f
28. 08100002
29. 31ce0000
30. ac0e3ff2
31. 2129ffff
32. 8d280000
33. 08100021
34. 31ad0000
35. 21ad0041
36. 150d0004
37. 31ce0000
38. 21ce08ee
39. ac0e3ff2
40. 08100002
41. 31ad0000
42. 21ad0042
43. 150d0004
44. 31ce0000
45. 21ce083e
46. ac0e3ff2
47. 08100002
48. 31ad0000
49. 21ad0043
50. 150d0004
51. 31ce0000
52. 21ce089c
53. ac0e3ff2
54. 08100002
55. 31ad0000
56. 21ad0044
57. 150d0004
58. 31ce0000
59. 21ce087a
60. ac0e3ff2
61. 08100002
62. 31ad0000
63. 21ad0045
64. 150d0004
65. 31ce0000
66. 21ce089e
67. ac0e3ff2
68. 08100002
69. 31ad0000
70. 21ad0046
71. 150d0004
72. 31ce0000
73. 21ce088e
74. ac0e3ff2
75. 08100002
76. 31ad0000
77. 21ad0047
78. 150d0004
79. 31ce0000
80. 21ce08f6
81. ac0e3ff2
82. 08100002
83. 08100002
84. 31ce0000
85. ac0e3ff2
86. 21290001
87. 8d280000
88. 08100021
89. 23bdfffe
90. afbf0001
91. afa40000
92. 1080000e
93. 8c88ffff
94. 21040000
95. 0c100058
96. 00000000
97. 8fbf0001
98. 8fa40000
99. 8c880000
100. ad280000
101. 21290001
102. 8c880001
103. 21040000
104. 0c100058
105. 00000000
106. 8fbf0001
107. 23bd0002
108. 03e00008
109. 33390000
110. 23390001
111. 0019cc80
112. 2339ffff
113. 1720fffe
114. 03e00008

(4)测试用例的运行结果





**五、设计遇到的问题及解决方法**



**六、设计总结及设计心得**

