一、

a)31 代表 31 号寄存器，即$ra,执行 jal 时以及 jalr $t1 或者 jalr $t1,$t2（$t1 默认为$ra）时，需要将 PC+4 保存到$ra。

b)对应的指令[10-6]是 shamt，执行 sll,srl,sra 指令时 ALUSrc1 需要置 1，因为需要读入位移量。

c)对应多路选择器选择写入寄存器的信号为 PC+4,执行 jal,jalr 时需要置 MemtoReg 为 2，因为此时需要将 PC+4 写入到$ra

d)对应多路选择器选择保存在寄存器中的地址，执行 jr,jalr 时需要置 PCSrc 为 2，因为此时需要读取保存在寄存器中的跳转目标的地址

e)可能进行符号位扩展也可能进行零扩展。进行符号位扩展时 Extop 为 1，如 lw,sw,addi,addiu,beq,slti,sltiu,进行 0 扩展时 Extop 为 0，如 andi

f)添加或门，输入为指令的所有位，当且仅当所有位为 0 时，输出为 0，设该信号为 Enable,对 MemRead,MemWrite,RegWrite 添加新的控制信号，输入为 Enable,当 Enable 为 1 时选择原来的 MemRead,MemWrite,RegWrite,当 Enable 为 0 时，全置 0，则什么也不做。

二、

1)

	PcSrc[1:0]	Branch	RegWrite	RegDst[1:0]	MemRead	MemWrite	MemtoReg[1:0]	ALUSrc1	ALUSrc2	Extop	Luop
lw	0	0	1	0	1	0	1	0	1	1	0
sw	0	0	0	x	0	1	x	0	1	1	0
lui	0	0	1	0	0	0	0	0	1	x	1
add	0	0	1	1	0	0	0	0	0	x	x
sub	0	0	1	1	0	0	0	0	0	x	x
subu	0	0	1	1	0	0	0	0	0	x	x
addi	0	0	1	0	0	0	0	0	1	1	0
addiu	0	0	1	0	0	0	0	0	1	1	0
and	0	0	1	1	0	0	0	0	0	x	x
or	0	0	1	1	0	0	0	0	0	x	x
xor	0	0	1	1	0	0	0	0	0	x	x
nor	0	0	1	1	0	0	0	0	0	x	x
andi	0	0	1	0	0	0	0	0	1	0	0
sll	0	0	1	1	0	0	0	1	0	x	x
srl	0	0	1	1	0	0	0	1	0	x	x
sra	0	0	1	1	0	0	0	1	0	x	x
slt	0	0	1	1	0	0	0	0	0	x	x
sltu	0	0	1	1	0	0	0	0	0	x	x
slti	0	0	1	0	0	0	0	0	1	1	0
sltiu	0	0	1	0	0	0	0	0	1	1	0
beq	0	1	0	x	0	0	x	0	0	1	0
j	1	x	0	x	0	0	x	x	x	x	x
jal	1	x	1	2	0	0	2	x	x	x	x
jr	2	x	0	x	0	0	x	x	x	x	x
jalr	2	x	1	2	0	0	2	x	x	x	x
addu	0	x	1	1	0	0	x	0	0	x	x

2)

足够长时间后，因为最后的 j loop 语句而发生死循环

$a0=0+12345=0x00003039,

$a1=-11215=0xffffd431,

$a2=-11215*$2^{16}$=0xd4310000，

$a3=0xd4310000/$2^{16}$=0xffffd431,

$t0=0xd4310000+0x00003039=0xd4313039，

$t1=0xd4313039/$2^{8}$=0xffd43130，

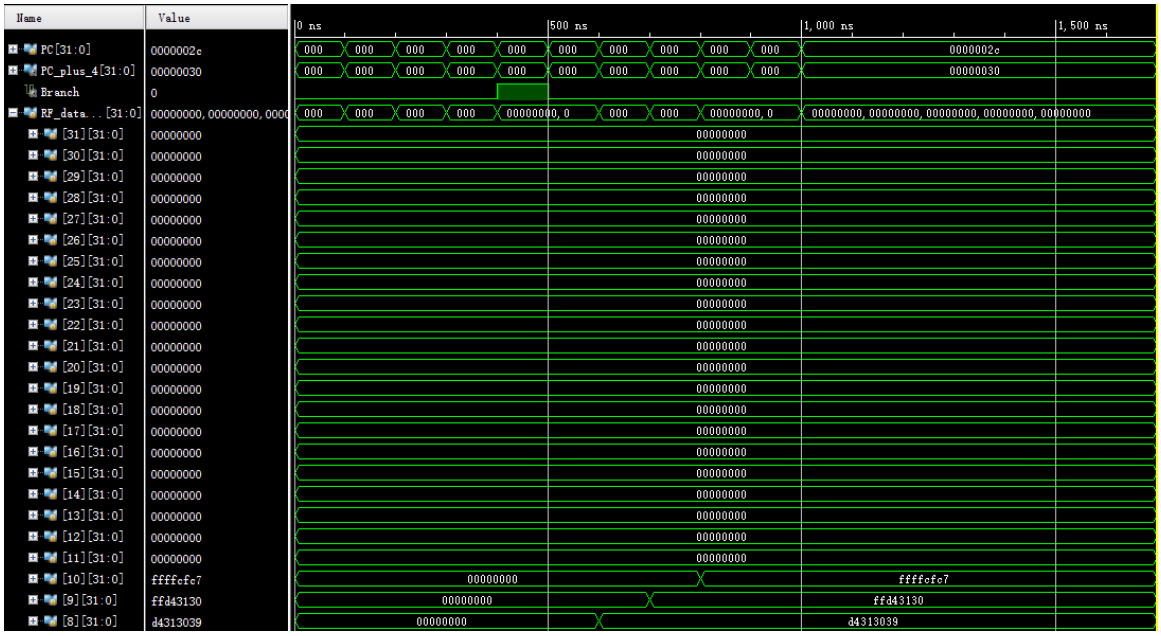
$t2=-12345=0xffffcfc7,

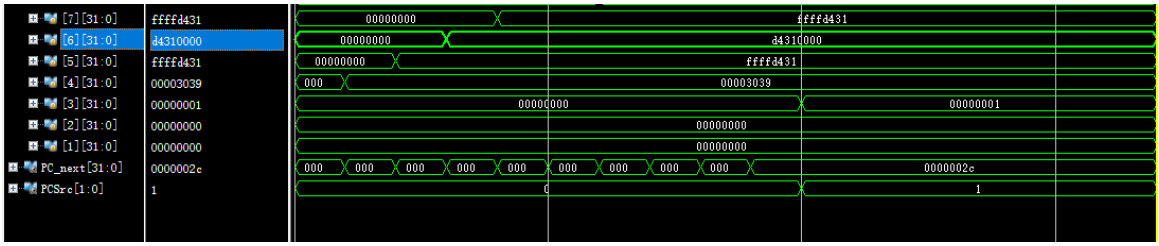
$v0=0,(有符号比较，$a0>0,$t2<0，则不满足$a0<$t2)

$v1=1。(无符号比较，将$t2 视为无符号的 ffffcfc7，即 $4294954951_{(10)}$，则满足$a0<$t2)

不能判断出是有符号还是无符号数，因为二者在计算机中存储表示方式完全相同，只有执行有符号、无符号命令时，才可人为地将其判定为有符号或无符号的数。

3）仿真结果如下图，共仿真 1500ns





a)PC 除了在第 500ns 时由 8’h00000010 变为 8’h0000018 外，每 100ns（一个周期）都加 4

b)Brach 信号在 400~500ns 为 1，它使得 PC 在 500ns 时由 8’h00000010 变为 8’h0000018

c)100~200ns 时，PC 值为 8’h00000004,对应 addiu $a1, $zero, -11215 指令，200~300ns，$a1 值为 8’hffffd431,,因为上一条指令将$a1 赋值为-11215 其补码表示为 8’hffffd431,不会出现错误，$a1 的值在上升沿就传入，而执行下一条指令取指等需要一定时间。

d)运行足够长时间后，

$a0 8’h00003039,$a1 8’hffffd431,$a2 8’hd4310000,$a3 8’hffffd431,

$t0 8'hd4313039,$t1 8'hffd43130,$t2 8'hfffffcfc7,$v0 8'h00000000,$v1 8'h00000001 都与预期一致

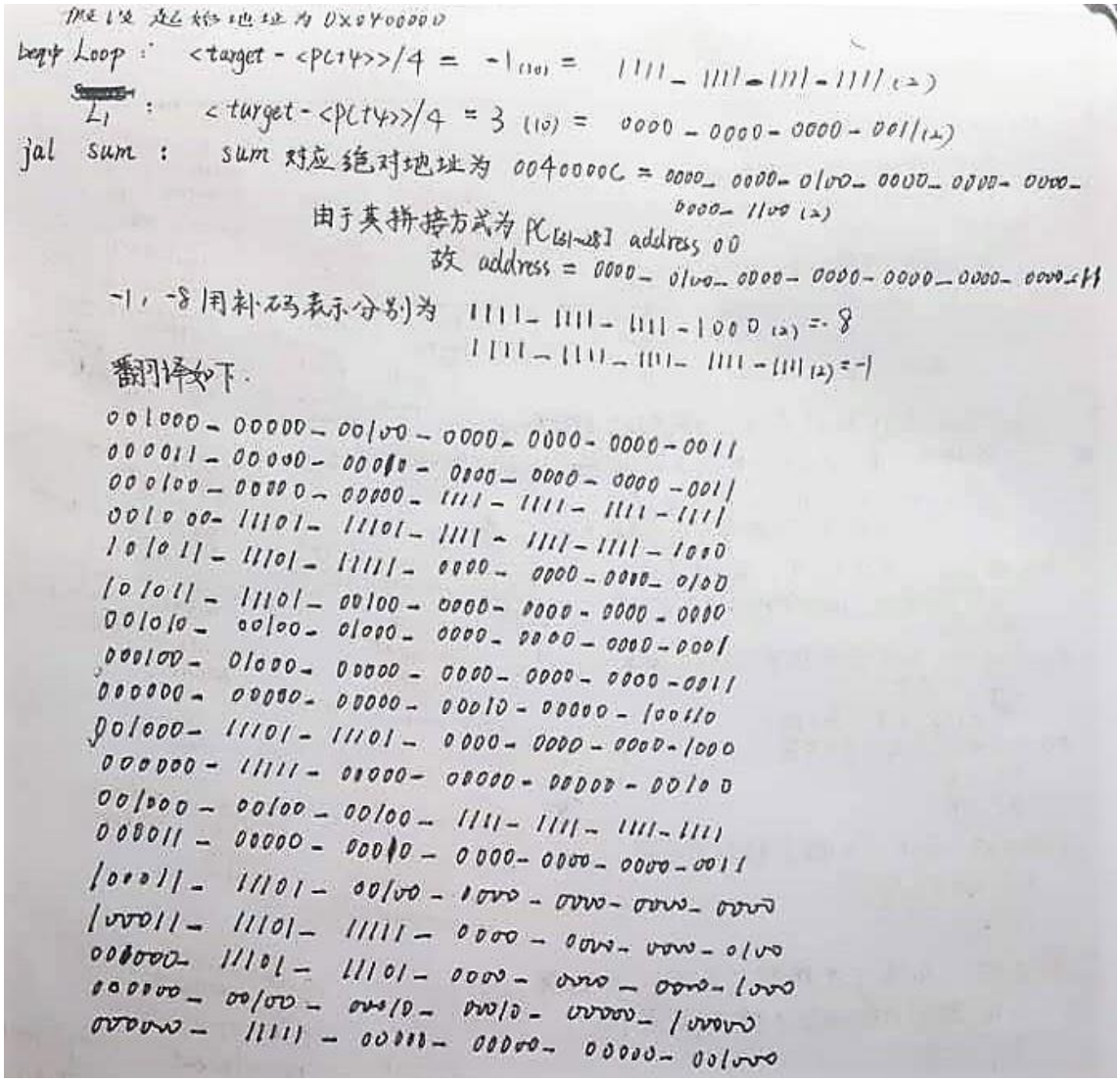
三、

1)

```
addi $a0,$zero,3#a0=3
jal sum#调到 sum,并将 pc+4 存入 ra
Loop:
  beq $zero, $zero, Loop#Loop 死循环
sum:
  addi $sp,$sp,-8#栈顶指针-8，为之后压栈做准备
  sw $ra,4($sp)#ra 中地址入栈
  sw $a0,0($sp)#当前 a0 的值入栈
  slti $t0,$a0,1#a0<1 时，t0=1,否则 t0=0
  beq $t0,$zero,L1#t0=0 时，跳转到 L1
  xor $v0,$zero,$zero#v0=0
  addi $sp,$sp,8#栈顶指针+8，为之后出栈做准备
  jr $ra#跳转到 ra 存储的地址
  L1:
  addi $a0,$a0,-1#a0-1
  jal sum#跳到 sum，并将 pc+4 存入 ra
  lw $a0,0($sp)#读取当前 L1 段父过程的 a0 值
  lw $ra,4($sp)#读取父过程未执行步骤的地址
  addi $sp,$sp,8#栈顶指针+8，为之后出栈做准备
  add $v0,$a0,$v0#累加 v0=a0+v0
  jr $ra#跳转到 ra 存储的地址
```

实现累加计算 1+2+…+n。Loop 计算完成后在原地循环。sum 实现对递归调用次数的控制，L1 减小递归问题中表示问题规模的 n 的数值，进而使递归是有限次内可完成的，同时完成对结果的累加。

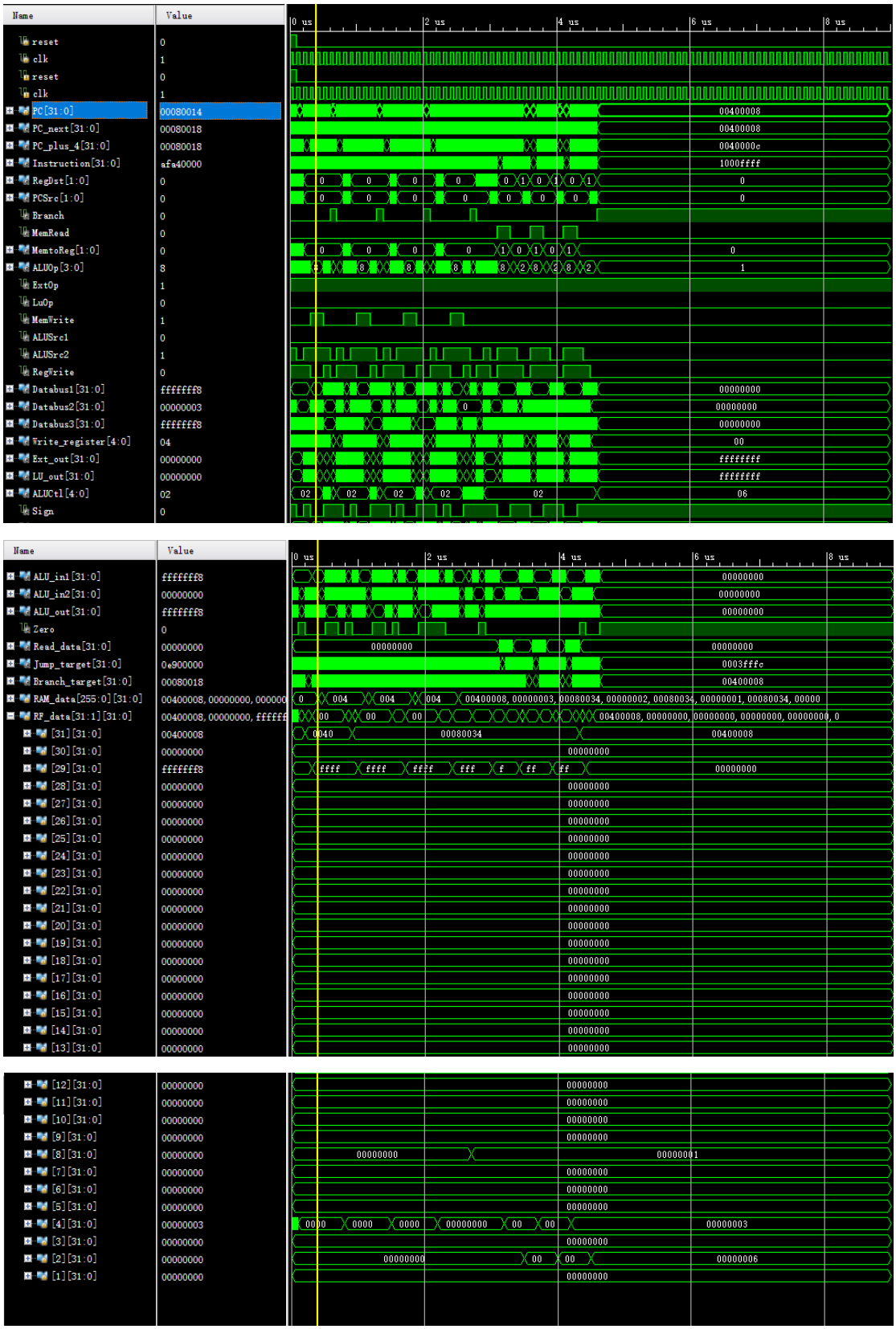
2)



3）

不仅需要修改 InstructionMemory 还需要修改 Control 模块，修改后的 CPU（假设起始地址为 32’h00400000,不是 32’h00000000）已经打包在作业文件内。

仿真结果如下，共仿真 9us。



a)$a0=3,$v0=6，和预期一致

b)PC 在每个上升沿改变，根据当前指令变为(PC+4)或指定地址。每个周期都会计算 PC+4、Branch_target、Jump_target 的值，根据需要 PC_next 选择相应的值。

$a0 变化规律为 3、2、1、0、1、2、3。$a0 是递归函数的局部变量，随着压栈（sw）、出栈

（lw）而发生变化。当$a0 未减小到 0（递归未到达终点）时,$a0 逐渐减小，随后累加时随着出栈而逐渐变大。

$v0 变化规律为 0、1、3、6，在递归结束，返回父过程时，$v0 作为累加变量保存最终求和结果。

$sp 是栈顶指针。当进入递归子过程时，$sp-8，为压栈做准备，当递归结束返回到父过程时，$sp+8 为出栈做准备。

$ra 是 jal 指令保存的应该执行的下一步指令的地址（PC+4）。只有在 jal 指令执行过后，$ra 才改变。