第 7 章　控制器作业+答案

7.8 如图 7.27 所示的单周期 CPU 数据通路，在其上执行下列 MIPS 指令，描述执行过程，并据此总结 ALU 必须具备的运算功能：

（1）　xor rd,rs,rt;　　位异或：rs⊕rt→rd

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 xor 的 OP 和 func，发送出 w_r_s=00，rt_imm_s=0，wr_data_s=00，ALU_OP 为异或的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端（因为 rt_imm_s=0），ALU 执行异或操作（ALU_OP 指定）；**
- **在 clk 的下跳沿，运算结果写入 rd 字段指定的寄存器（因为 w_r_s=00、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（2）　sltu rd,rs,rt;　　无符号数小于则置位：if (rs < rt) rd=1 else rd=0

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 sltu 的 OP 和 func，发送出 w_r_s=00，rt_imm_s=0，wr_data_s=00，ALU_OP 为小于置位操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端（因为 rt_imm_s=0），ALU 执行小于置位操作（ALU_OP 指定）；在 clk 的下跳沿，运算结果写入 rd 字段指定的寄存器（因为 w_r_s=00、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（3）　sllv rd,rt,rs;　　逻辑左移：(rt << rs)→rd

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 sllv 的 OP 和 func，发送出 w_r_s=00，rt_imm_s=0，wr_data_s=00，ALU_OP 为逻辑左移操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端（因为 rt_imm_s=0），ALU 执行逻辑左移操作（ALU_OP 指定）；**
- **在 clk 的下跳沿，运算结果写入 rd 字段指定的寄存器（因为 w_r_s=00、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（4）　andi rt, rs, imm;　　逻辑与：rs & imm→rt

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**

- **“译码及控制单元”依据 andi 的 OP 字段，发送出 w_r_s=01，imm_s=0，rt_imm_s=1，wr_data_s=00，ALU_OP 为逻辑与操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段送寄存器堆的 A 口地址，rt 字段送寄存器堆的写端口地址（因为 w_r_s=01），A 口读出数据送 ALU 的 A 端，操作码的低 16 位经过无符号扩展（因为 imm_s=0）后的 32 位数据送 ALU 的 B 端（因为 rt_imm_s=1），ALU 执行逻辑与操作（ALU_OP 指定）；**
- **在 clk 的下跳沿，运算结果写入 rt 字段指定的寄存器（因为 w_r_s=01、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（5） sw rt, offset(rs)； 存数：rt →（rs +offset）

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 sw 的 OP 字段，发送出 imm_s=1，rt_imm_s=1，ALU_OP 为算术加操作的运算码，Write_Reg=0，Mem_Write=1，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，A 口读出数据送 ALU 的 A 端，操作码的低 16 位经过符号扩展（因为 imm_s=1）后的 32 位数据送 ALU 的 B 端（因为 rt_imm_s=1），ALU 执行逻辑加操作（ALU_OP 指定）计算的地址直接送数据存储器的地址端；**
- **在 clk 的下跳沿，寄存器堆的 B 口数据（即 rt 寄存器数据）写入数据存储器指定的单元（因为 Mem_Write=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（6） bne rs, rt, label； 不相等转移：if(rs≠rt) PC+4+offset*4→PC

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 bne 的 OP 字段，发送出 imm_s=1，rt_imm_s=0，ALU_OP 为算术减操作的运算码，Write_Reg=0，Mem_Write=0，PC_s=10（ZF=0）或者 PC_s=00（ZF=1）；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端，进行算术减法操作；同时，操作码的低 16 位经过符号扩展（因为 imm_s=1）后的 32 位数据送相对地址加法器，进行左移 2 位操作并与 PC+4 值进行相加，结果备用；**
- **在 clk 的下跳沿，根据 ALU 减法运算的结果修改 PC：ZF=0 则将相对地址加法器结果置入 PC（因为 PC_s=10），ZF=1 则将 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

7.9 在图 7.27 所示的单周期 CPU 数据通路上，假设 ALU 的功能及 ALU_OP 编码如下：

ALU_OP	操作
000	算术加
001	算术减

010	位异或
011	逻辑左移
100	小于置位

在其上为实现下列 MIPS 指令，写出译码与控制单元所需设置的控制信号，填入下表：

（1） xor rd,rs,rt; 位异或：rs⊕rt→rd

（2） sltu rd,rs,rt; 无符号数小于则置位：if (rs < rt) rd=1 else rd=0

（3） sllv rd,rt,rs; 逻辑左移：(rt << rs)→rd

（4） xori rt, rs, imm; 逻辑与：rs & imm→rt

（5） sw rt, offset(rs); 存数：rt →（rs +offset）

（6） bne rs, rt, label; 不相等转移：if(rs≠rt) PC+4+offset*4→PC

（7） jal label; 无条件跳转并链接：(PC+4)→$31, {(PC+4) 高 4 位,address,0,0}→PC

指令	w_r_s	imm_s	rt_imm_s	wr_data_s	ALU_OP	Write_Reg	Mem_Write	PC_s
xor rd,rs,rt	00	—	0	00	010	1	0	00
sltu rd,rs,rt	00	—	0	00	100	1	0	00
sllv rd,rt,rs	00	—	0	00	011	1	0	00
xori rt, rs, imm	01	0	1	00	010	1	0	00
sw rt, offset(rs)	—	1	1	—	000	0	1	00
bne rs, rt, label	—	1	0	—	001	0	0	ZF=0:10 ZF=1:00
jal label	10	—	—	10	—	1	0	11

7.10 伪指令是指机器指令系统中不存在，但是汇编器能将其翻译为某一条或者多条机器指令来实现的符号指令。譬如，移动指令 move rd,rs 是将 rs 内容传送给 rd 寄存器的伪指令，汇编器可以将它翻译为 add rd,rs,$0。MIPS 中有一条取立即数伪指令 li rd,imm，功能是将 16 位立即数 imm 存入 rd 寄存器（高 16 位清零）。

（1） 结合 MIPS 的核心指令集，思考：这条伪指令会被 编译器翻译成哪一条或哪几条指令？

可以被编译器翻译成以下任何一条指令：

ori rd, $zero, imm

xori rd, $zero, imm

（2） 假设现在要直接在 MIPS CPU（单周期）指令系统中加入这条指令（不是伪指令），那么能否在图 7-27 所示的单周期 CPU 数据通路上实现？如果不能，那么需要添加哪些其他部件和数据通路才能实现？请画出并说明执行过程。

可以实现，两种方法：

一种方法可以直接在 li 指令编码时，规定其 rs 字段一定为 00000，即指定 rs 字段为$0，这种方法无须添加部件；另一种方法是：在寄存器堆的 A 地址端口前添加一个 5 位的二选一电路，由 rA_s 信号控制，rA_s=0，则选择指令码的 rs 字段送寄存

器的 A 口地址，rA_s =1，则选择 00000 送寄存器的 A 口地址；在执行 li 指令时，发送 rA_s =1，其他指令时，发送 rA_s =0。

7.11 假设要实现一条 lw_inc rt,offset(rs)指令，其功能是在普通的取数指令 lw 的功能基础上，增加递增功能，即取数后，递增地址索引寄存器 rs 的值。lw_inc 相当于执行了两条指令：

lw rt,offset(rs)

addi rs,rs,1

在图 7-27 所示的单周期 CPU 数据通路上添加一些部件和控制信号，实现该指令，说明你的设计方案和指令执行过程。

设计方案：可以强化寄存器堆的功能，使其能对 A 口地址指明的寄存器执行+1 功能，这要求寄存器堆中每一个寄存器都有+1 计数功能，寄存器堆添加一个控制信号 rA_1，当该信号=1 时，在 clk 的下跳沿，将 A 口地址选中的寄存器内容+1。

执行过程：在时钟周期上跳沿，根据 PC 从指令存储器读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；“译码及控制单元”依据 lw+的 OP 字段，发送出 w_r_s=01，imm_s=1，rt_imm_s=1，wr_data_s=01，ALU_OP 为算术加操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00，rA_1=1；指令码的 rs 字段送寄存器堆的 A 口地址，rt 字段送寄存器堆的写端口地址（因为 w_r_s=01），A 口读出数据送 ALU 的 A 端，操作码的低 16 位经过符号扩展（因为 imm_s=1）后的 32 位数据送 ALU 的 B 端（因为 rt_imm_s=1），ALU 执行逻辑加操作（ALU_OP 指定），计算的地址直接送数据存储器的地址端；在 clk 的下跳沿，数据存储器读出的数据写入 rt 指定的寄存器（因为 w_r_s=01，wr_data_s=01 和 Mem_Write=0），且 rs 指定的寄存器执行+1 操作（因为 rA_1=1），PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。

7.12 在图 7-27 所示的单周期 CPU 数据通路上，假设多路复用器的下列选择控制信号发生了恒零错误（即始终为 0），考虑：哪些指令可以正常工作，哪些指令不可以正常工作？

（1） w_r_s=00b;

可以正常工作的指令：R 型、I 型分支类指令、j 指令、sw 指令；

不能正常工作的指令：I 型运算类指令、lw 指令、jal 指令，因为无法写 rt 或者 $31；

（2） imm_s=0b;

可以正常工作的指令：R 型、J 型；I 型无符号运算类指令及逻辑运算类指令

不能正常工作的指令：I 型有符号运算类以及分支指令、sw 和 lw 指令，因为无法符号扩展；

（3） wr_data_s=00b;

可以正常工作的指令：R 型、I 型分支类指令、I 型运算类指令、j 指令、sw 指令；

不能正常工作的指令： lw 指令、jal 指令，因为无法将存储器读出数据或者 PC+4 的值写入寄存器堆；

（4） rt_imm_s=0b;

可以正常工作的指令：R 型、J 型指令、I 型分支指令；

不能正常工作的指令： I 型运算类指令、I 型访存类指令，因为无法将立即数字段送到 ALU 的 B 端；

（5） PC_s=00b;

可以正常工作的指令：R 型、I 型运算类指令、I 型访存指令；

不能正常工作的指令：转移类指令，包括 jr、j、jal、I 型分支类指令，因为无法更新 PC 为其他的值（除 PC+4 外）；

7.13 在图 7-27 所示的单周期 CPU 数据通路上，考虑多路复用器的选择控制信号发生了恒 1 错误对 CPU 的影响。考虑：哪些指令可以正常工作，哪些指令不可以正常工作？

（1） w_r_s=11b;

可以正常工作的指令：I 型分支类指令、jr 指令、sw 指令；

不能正常工作的指令：R 型、I 型运算类指令、lw 指令、jal 指令，因为无法写 rd、rt 或者$31；

（2） imm_s=1b;

可以正常工作的指令：R 型、J 型；I 型有符号运算类指令、以及分支指令、sw 和 lw 指令；

不能正常工作的指令：I 型无符号运算类或者逻辑运算类，因为无法进行无符号扩展；

（3） wr_data_s=11b;

可以正常工作的指令：I 型分支类指令、jr 指令、sw 指令；

不能正常工作的指令：R 型、I 型运算类指令、lw 指令、jal 指令，因为无法将 ALU 运算结果、或者存储器读出数据或者 PC+4 的值写入寄存器堆；

（4） rt_imm_s=1b;

可以正常工作的指令：I 型运算类指令、I 型访存指令、jr、j、jal 指令；

不能正常工作的指令：R 型、I 型分支类指令，因为无法将 rt 的内容送到 ALU 的 B 端；

（5） PC_s=11b;

可以正常工作的指令：j、jal 指令；

不能正常工作的指令：其他指令，因为无法更新 PC 为其他的值（除页面寻址的转移地址之外）；

7.14 在图 7-42 所示的多周期 CPU 的数据通路上，描述下列指令的执行步骤及每个机器周期中需要发送的操作控制信号序列。假设 ALU 的功能及 ALU_OP 编码如下：

ALU_OP	操作
100	位与
101	位或
000	位异或
001	位非
010	算术加

011	算术减
110	小于置位
111	逻辑左移

（1） xor rd,rs,rt；位异或：rs⊕rt→rd

（2） sltu rd,rs,rt；无符号数小于则置位：if (rs < rt) rd=1 else rd=0

（3） sllv rd,rt,rs；逻辑左移：(rt << rs)→rd

（4） andi rt, rs, imm；逻辑与：rs & imm→rt

（5） bne rs, rt, label；不相等转移：if(rs≠rt) PC+4+offset*4→PC

时钟周期	操作	发送控制信号
（1）xor rd,rs,rt		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）B→F	ALU_A_s=1, ALU_B_s=01, ALU_OP=000;
M3	F→ Reg[rd]	rd_rt_s=0,alu_mem_s=0, Reg_write;
（2）sltu rd,rs,rt		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）B→F	ALU_A_s=1, ALU_B_s=01, ALU_OP=110;
M3	F→ Reg[rd]	rd_rt_s=0,alu_mem_s=0, Reg_write;
（3）sllv rd,rt,rs		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）B→F	ALU_A_s=1, ALU_B_s=01, ALU_OP=111;
M3	F→ Reg[rd]	rd_rt_s=0,alu_mem_s=0, Reg_write;
（4）andi rt, rs, imm		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;

时钟周期	操作	发送控制信号
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）imm→F	ALU_A_s=1, ALU_B_s=10, imm_s=0,ALU_OP=100;
M3	F→ Reg[rt]	rd_rt_s=1,alu_mem_s=0, Reg_write;
（5）I 型分支指令：beq		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0, Mem_read, IR_write; ALU_OP=010; ALU_A_s=0,ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B PC+offset*4→F	ALU_A_s=0,ALU_B_s=11,ALU_OP=010,imm_s=1;
M2	A - B ，产生 zero zero=1，则 F→PC zero=0，空操作	ALU_A_s=1, ALU_B_s=01, ALU_OP=011; zero=1:PC_s=01,PC_write

7.15 综合上题中 5 条指令的操作控制信号的逻辑函数，画出其硬布线控制器的电路图。

（1）综合微操作控制信号的函数：

I_D_s = 0

Mem_read = M0

Mem_write = 0

IR_write = M0

PC_write = M0+ beq • M2 • zero

Reg_write = （~beq）• M3

imm_s= beq • M1

ALU_A_s = M2

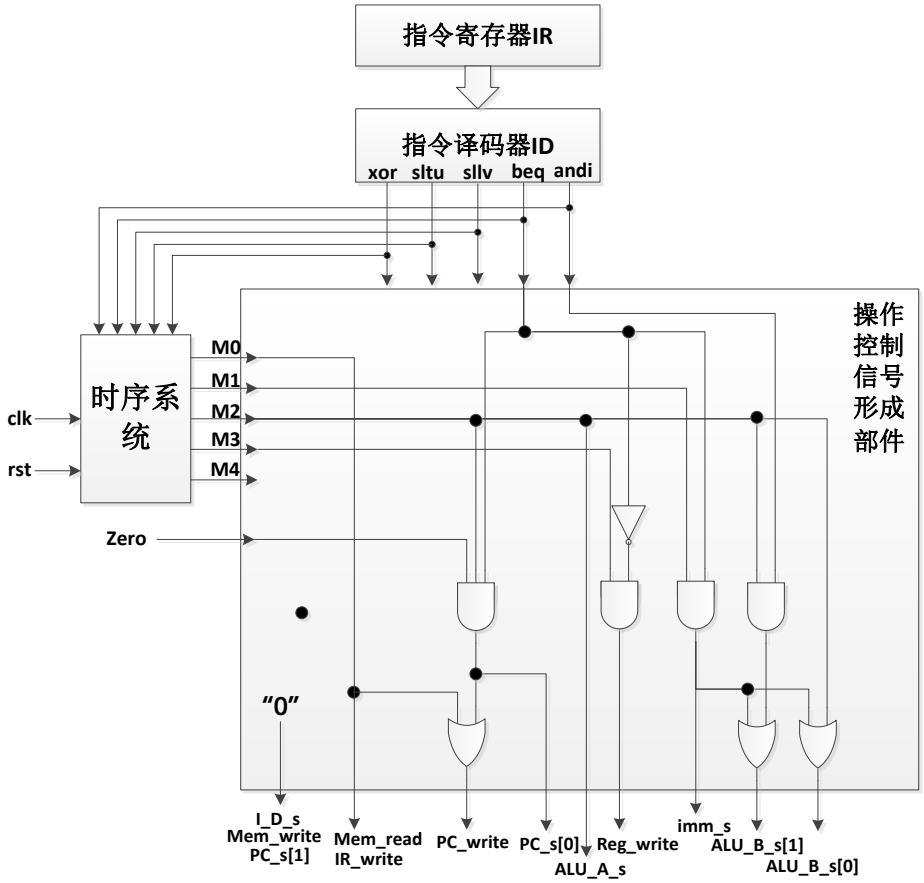
ALU_B_s[1] = andi • M2 + beq • M1

ALU_B_s[0] = M2 + beq • M1

PC_s[1] = 0

PC_s[0] = beq • M2 • zero

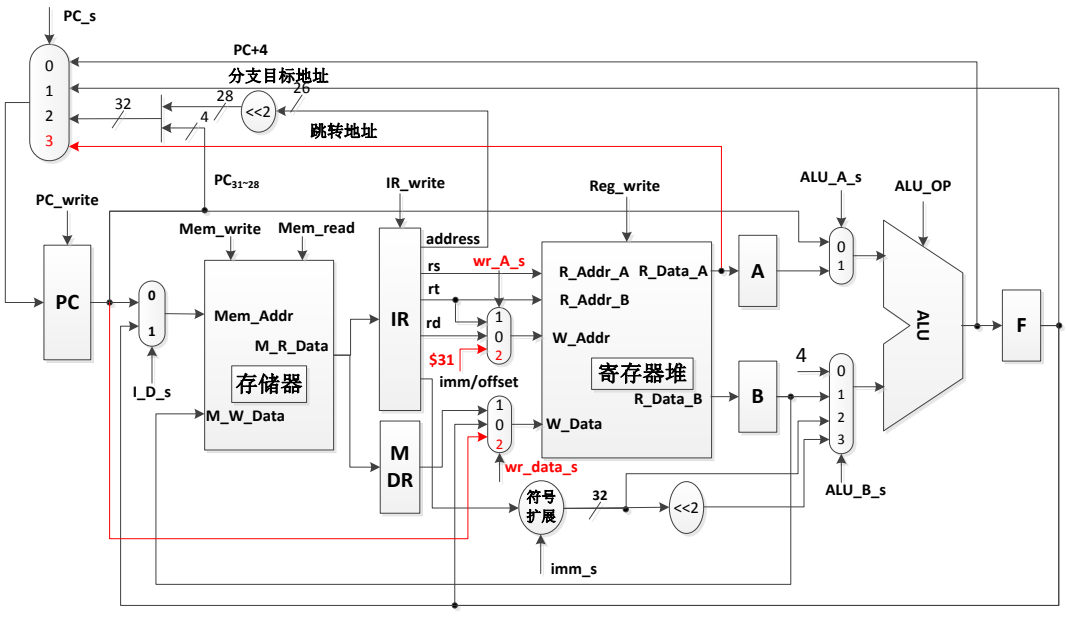
（2）画出电路图：



7.16 假如要在图 7-42 所示的多周期 CPU 的数据通路上，实现以下两条转移指令，请分析是否可行？如果不行，请尝试增加部件或者更改数据通路，以实现以下两条指令；阐述你的方案，画出数据通路图，说明指令执行过程。

（1） jr rs；无条件跳转：rs → PC

（2） jal label；无条件跳转并链接：(PC+4)→$31, {(PC+4)高 4 位,address,0,0}→PC



（1） 为实现 jr rs 指令，可以添加一条链路、在 PC 置位的多路选择器中添加一个选项，变成 4 选 1，当 PC_s=11 时，将寄存器堆的 R_data_A 端口数据送到多路选择器的选项 4 处。

（2） 为实现 jal label 指令，则需要在寄存器写端口的地址和数据端的多路选择器多一个选项，分别是$31 和 PC+4，也即 3 选 1。控制信号改为 wr_A_s 和 wr_data_s。指令执行过程：

时钟周期	操作	发送控制信号
（1）jr rs		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=+; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→PC	PC_s=11, PC_write;
（2）jal Label		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=+; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	PC+4→reg[$31] {(PC+4)高 4 位,add-ress,0,0}→PC	wr_A_s=10, wr_data_s=10, Reg_write PC_s=10, PC_write;

7.17 MIPS 为 RISC CPU，指令集相对简单，实现复杂指令的方法是将其用多条简单指令来代替。思考 MIPS 如何实现交换指令 swap rs,rt（将寄存器$rs 和$rt 中的内容交换）？可以使用多条简单指令实现，也可以以单周期和多周期 CPU 的方式实现，同时考虑允许有另一个寄存器内容被改变和不允许其他寄存器内容被改变两种情况。

（1） 用三条简单指令实现：

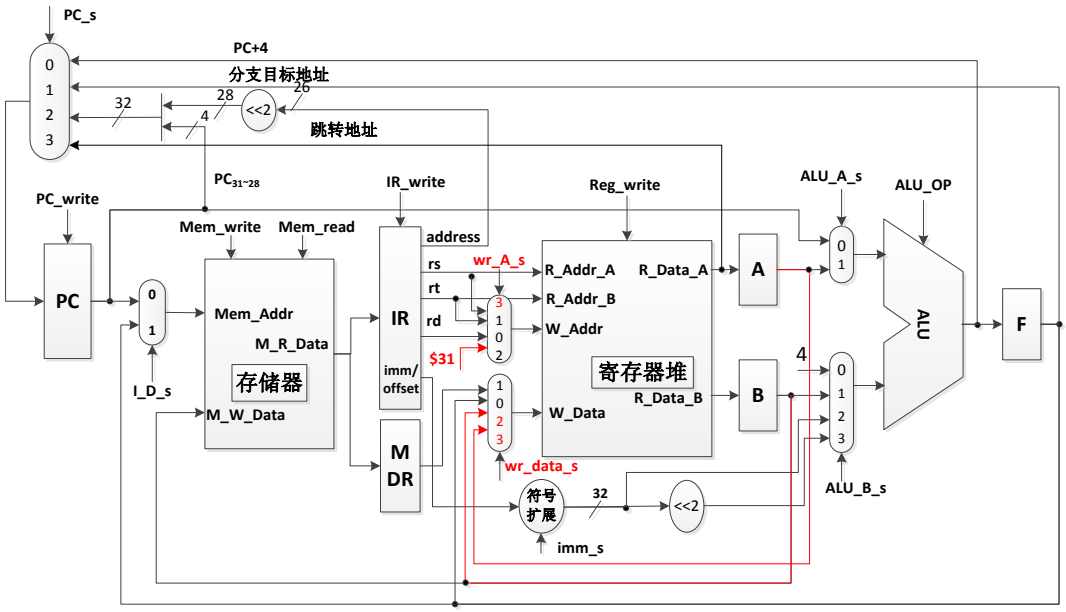
使用汇编器保留的寄存器$1($at)做交换媒介：

```
xor   $1,$zero,$rt      ; rt→$1
xori  rt,rs,0x0000      ; rs⊕0→rt
xor   $rd,$zero, $1     ;  $1⊕0→rd，此处 rd 的编码为 rs
```

（2） 使用多周期 CPU 实现：

不用改变另外一个寄存器的内容；

添加两个数据通路：寄存器写地址端添加一个来源，即指令码的 rs 字段；寄存器写数据端也添加两个来源，即暂存器 A 的输出和暂存器 B 的输出。



时钟周期	操作	发送控制信号
swap rs,rt		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=+; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A, Reg[rt]→B	
M2	A→Reg[rt]	wr_A_s=01, wr_data_s=11,Reg_write;
M3	B→Reg[rs]	wr_A_s=11, wr_data_s=10,Reg_write;

（3） 如果使用单周期 CPU 在一个时钟周期中实现数据交换，那么需要将寄存器堆的写端口修改成双端口写的结构，略。