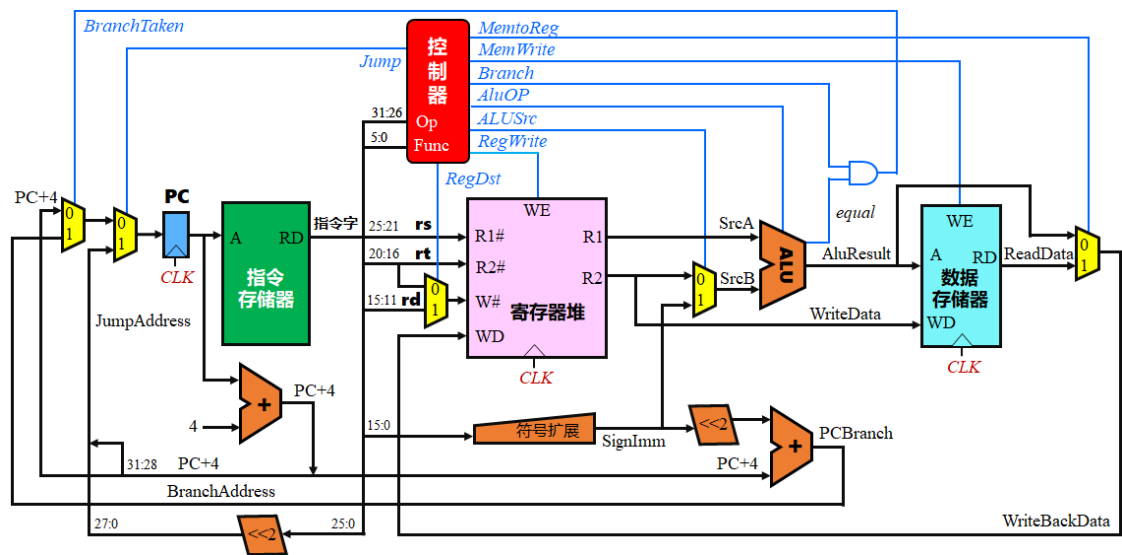


MIPS单周期可执行24条指令CPU

实验要求

本实训项目帮助学生构建支持 24 条指令的 MIPS 单周期 CPU，最终实现的处理器能运行 benchmark 测试程序。另外希望学有余力的同学能为自己的处理器增加中断处理机制，能响应外部设备中断请求。



CPU基本框架

指令格式

R	opcode	rs	rt	rd	shamt	funct
	31	26 25	21 20	16 15	11 10	6 5
						0
I	opcode	rs	rt	immediate		
	31	26 25	21 20	16 15		0
J	opcode	address				
	31	26 25				0

指令格式

24条指令

指令类型	指令	助记符
R型	移位运算	SLL、SRA、SRL
	算数运算	ADD、ADDU、SUB
	逻辑运算	AND、OR、NOR
	比较运算	SLT、SLTU
	分支	JR *
	系统调用	SYSCALL *
I型	分支	BEQ、BNE
	立即数运算	ADDI、ADDIU、SLTI、ANDI、ORI
	访问内存	LW、SW
J型	跳转	J、JAL

处理器应支持的指令

*为特殊格式指令

指令详解

R型

名称	助记符	操作
Shift Left Logical	sll	$R[rd] = R[rt] \ll \text{shamt}$
Shift Left Arithmetic	sra	$R[rd] = R[rt] \gg \text{shamt}$
Shift Right Logical	srl	$R[rd] = R[rt] \gg \text{shamt}$
Add	add	$R[rd] = R[rs] + R[rt]$
Add Unsigned	addu	$R[rd] = R[rs] + R[rt]$
Subtract	sub	$R[rd] = R[rs] - R[rt]$
And	and	$R[rd] = R[rs] \& R[rt]$
Or	or	$R[rd] = R[rs] R[rt]$
Nor	nor	$R[rd] = \sim (R[rs] R[rt])$
Set Less Than	slt	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$
Set Less Than Unsigned	sltu	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$
Jump Register	jr	$PC = R[rs]$
System Call	syscall	SignalException

I型

名称	助记符	操作
Branch On Equal	beq	if(R[rs] == R[rt]) then PC = PC + 4 + (SignExtImm << 2)
Branch On Not Equal	bne	if(R[rs] != R[rt]) then PC = PC + 4 + (SignExtImm << 2)
Add Immediate	addi	R[rt] = R[rs] + SignExtImm
Add Immediate Unsigned	addiu	R[rt] = R[rs] + SignExtImm
Set Less Than Imm.	slti	R[rt] = (R[rs] < SignExtImm)? 1 : 0
And Immediate	andi	R[rt] = R[rs] & ZeroExtImm
Or Immediate	ori	R[rt] = R[rs] ZeroExtImm
Load Word	lw	R[rt] = Mem[R[rs]+SignExtImm]
Store Word	sw	Mem[R[rs]+SignExtImm] = R[rt]

J型

名称	助记符	操作
Jump	j	PC = JumpAddr
Jump And Link	jal	R[ra] = PC+4, PC = JumpAddr

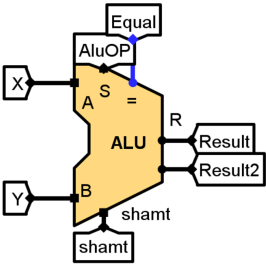
其中：

JumpAddr	(PC + 4) _{High4}	address	0
	31 28 27		2 1 0

```
JumpAddr = ((PC + 4) & 0xf0000000) | ((address << 2) & 0xffffffc);
```

设计硬布线

ALU结构及规格



ALU OP	Decimal	Function
0000	0	Result = X << Y 逻辑左移 (Y取低五位), Result2=0
0001	1	Result = X >>> Y 算术右移 (Y取低五位) Result2=0
0010	2	Result = X >> Y 逻辑右移 (Y取低五位) Result2=0
0011	3	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 无符号乘法
0100	4	Result = X/Y; Result2 = X%Y 无符号除法
0101	5	Result = X + Y (Set OF/UOF)
0110	6	Result = X - Y (Set OF/UOF)
0111	7	Result = X & Y 按位与
1000	8	Result = X Y 按位或
1001	9	Result = X ⊕ Y 按位异或
1010	10	Result = ~(X Y) 按位或非
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

```
Equal = (X == Y)?(1:0);
```

控制信号

控制信号产生条件

控制信号	说明	产生条件 (=1)
RegWrite	寄存器写使能	寄存器写入
MemWrite	内存写使能	sw
AluOP	运算器操作控制符 (4位)	R型指令依据 funct 字段
MemToReg	寄存器写入数据来自主存	lw
RegDst	写入寄存器编号选择	R型指令
AluSrcB	运算器B端输入选择	lw、sw、立即数运算指令
SignedExt	立即数符号扩展	addi、addiu、slti
JR	寄存器跳转指令信号	jr
JAL	JAL指令信号	jal
JMP	无条件转移控制信号	j、jal、jr
Beq	Beq指令信号	beq
Bne	Bne指令信号	bne
Syscall	系统调用指令信号	syscall

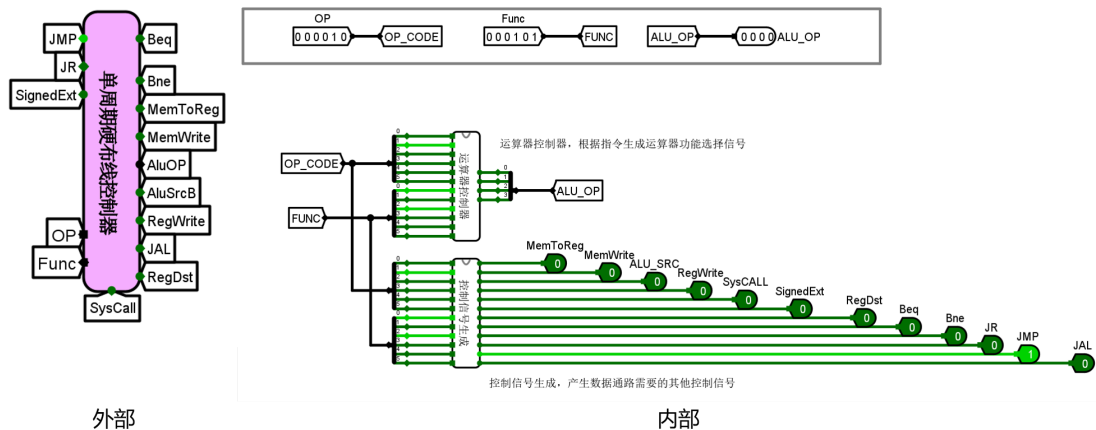
ALU_OP及控制信号的确定

根据 ALU 规格及控制信号产生条件即可得到下表，其中 X 代表无操作

OP	opcode	funct	ALU_OP	MemToReg	MemWrite	ALU_SrcB	RegWrite	Syscall	SignedExt	RegDst	BEQ	BNE	JR	JMP	JAL
sll	0	0	0				1			1					
sra	0	3	1				1			1					
srl	0	2	2				1			1					
add	0	32	5				1			1					
addu	0	33	5				1			1					
sub	0	34	6				1			1					
and	0	36	7				1			1					
or	0	37	8				1			1					
nor	0	39	10				1			1					
slt	0	42	11				1			1					
sltu	0	43	12				1			1					
jr	0	8	x							1			1	1	
syscall	0	12	x					1							
j	2	x	x											1	
jal	3	x	x				1							1	1
beq	4	x	x								1				
bne	5	x	x									1			
addi	8	x	5			1	1		1						
andi	12	x	7			1	1								
addiu	9	x	5			1	1		1						
slti	10	x	11			1	1		1						
ori	13	x	8			1	1								
lw	35	x	5	1		1	1		1						
sw	43	x	5		1	1			1						

ALU_OP及控制信号

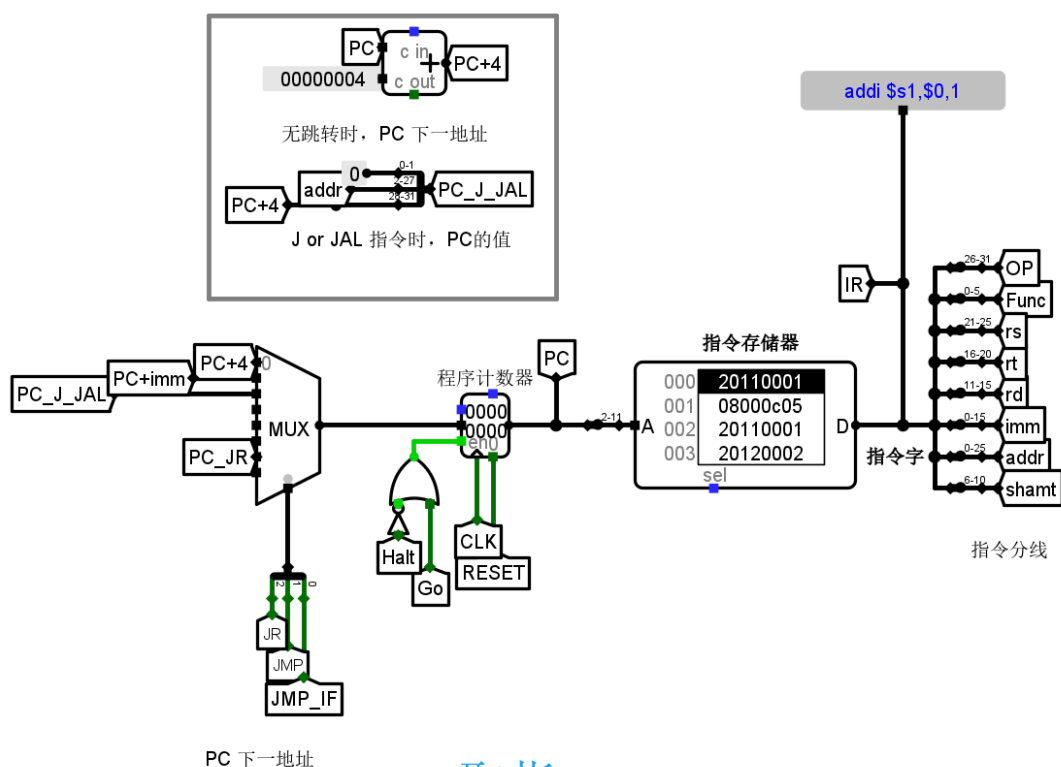
根据 Logisim 分析组合逻辑电路功能可以自动生成控制器内部逻辑



设计数据通路

因为是单周期CPU，所有指令均在一个时钟周期内完成操作，所以指令与数据分别采用ROM及RAM存储

• 取指



指令存储器 ROM 容量为 **1K * 32 bit**

由于指令存储器存储字长为 **4B**，而 PC 为指令字节地址，非指令字地址，故舍弃**最低两位**，将 2 ~ 11 位送入存储器地址端

指令 IR 分线后送入**单周期硬布线控制器**译码，同时下一指令的地址通过**多路选择器**送入程序计数器的数据端口，等待时钟上升沿锁存

PC下地址可通过下图产生，其中 X 代表该选择端无效：

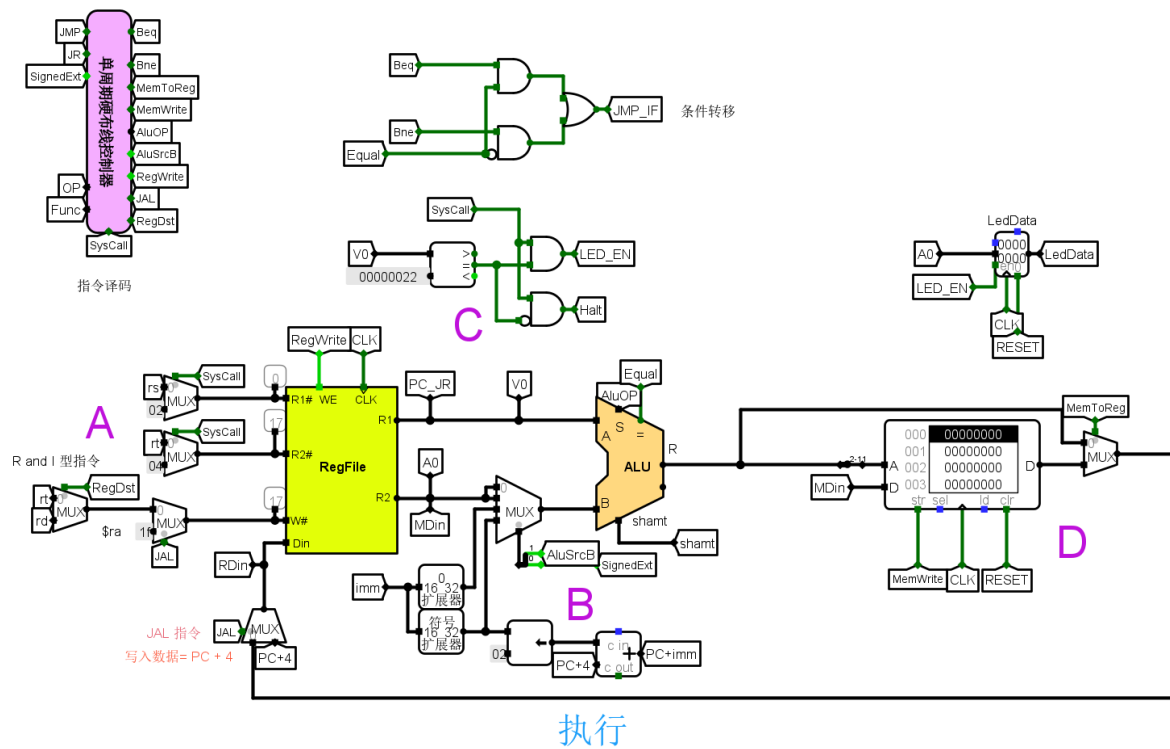
JR	JMP	JMP_IF	PC_Next
0	0	0	PC + 4
0	0	1	PC + imm
0	1	0	PC_J_JAL
0	1	1	X
1	0	0	X
1	0	1	X
1	1	0	PC_JR
1	1	1	X

JMP_IF 为**条件分支跳转**成功信号，只响应 Beq 或 Bne 指令：

$$JMP_IF = BeqEqual + Bne\overline{Equal}$$

```
PC = PC + 4; // PC + 4，无任何
跳转
PC = PC + 4 + (SignExtImm << 2); // PC + imm，当
Beq 或 Bne 指令条件有效
PC = ((PC + 4) & 0xf0000000) | ((address << 2) & 0x0ffffffc); // PC_J_JAL， J
或者 Jal 指令
PC = Reg[rs]; // PC_JR， JR指令
```

• 执行



Syscall 为系统调用指令，在该实验中，需要根据条件输出通用寄存器的值或产生停机信号：

```
if ($v0 == 0x22)
    /* 若通用寄存器 $v0 的值为 0x22，则输出 LedData,而 LedData 由寄存器锁存通用寄存器 $a0 的值 */
    std::cout >> LedData;
else{
    /* 否则，产生停机信号，程序不向下执行，直到 Go 按钮事件产生 */
    do{
        ;
    }while(Go != true );
}
```

1. A

通用寄存器 \$V0 编号为 2，\$a0 编号为 4，执行 syscall 指令时，**多路选择器**选择该组信号，其余指令选择 rs,rt

通用寄存器 \$ra 编号为 0x1f，记录返回地址，jal 指令可类比 x86 指令集中的 call 指令，所以该指令执行时，**多路选择器**选择 1 端口，写入**寄存器 W#** 选择 ra，将返回地址 **PC + 4** 写入寄存器文件 (RegFile)

2. B

根据控制信号 AluSrcB 及 SignedExt 作为**多路选择器**选择端，算逻运算单元 ALU 的 B 端输入数据如下图：

AluSrcB	SignedExt	B
0	0	R2
0	1	X
1	0	ZeroExtImm
1	1	SignExtImm

ALU_B 端数据

另外, $(PC + imm)$ 即为上文的具体实现

3. C

JMP_IF 信号上文已叙述

PC_JR 为 JR 指令执行时, 指令中通用寄存器 rs 的值, 该值作为新的 PC

- 执行 **syscall** 指令且通用寄存器 \$v0 值不为 **0x22** 时, 使停机信号 Halt 置 1, 该信号经**非门**使**程序计数器使能端**置 0, 将其封锁
- 若 \$v0 值为 0x22, 则 LED 更换数据信号 LED_EN 置 1, 在下一个时钟的上升沿将通用寄存器 \$a0 值锁存至 LedData 寄存器中

4. D

数据存储器 RAM 的容量与指令 ROM 一致, 均为 **1K * 32 bit**

本实验唯一用到数据存储器 RAM 的指令只有 **lw** 及 **sw**, 地址线依旧舍弃最低**两位**

MDin 为寄存器文件输出端 **R2** 的值, 在 sw 指令执行时, 通过 MemWrite 信号, 在时钟上升沿将其写入 RAM 中

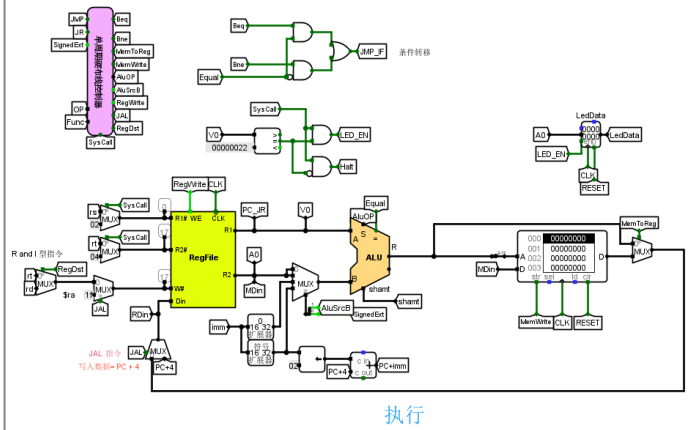
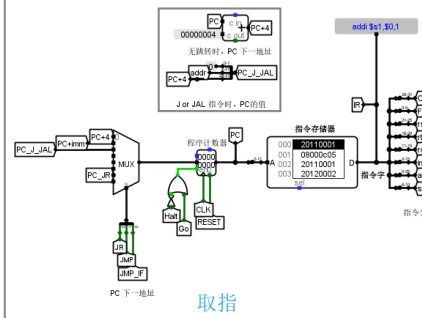
与此类似, lw 指令执行时, **多路选择器**选择内存数据, 送入寄存器文件 Din 端口, 在时钟上升沿写入 RegFile

备注

CLK 为时钟信号, RESET 为测试时清空各寄存器及 RAM 信号

最后完成的单周期 CPU

CPU框架



CPU