BEQ：相等时转移

编码	31 26	25 21	20 16	15 0	
	beq 000100	rs	rt	offset	
	6	5	5	16	
格式	beq rs, rt, offset				
描述	if (GPR[rs] == GPR[rt]) then 转移				

操作	if (GPR[rs] == GPR[rt])
	PC ← PC + 4 + sign_extend(offset‖0^2)
	else
	PC ← PC + 4				

示例	beq $s1, $s2, -2				
其他					

BGTZ：大于 0 时转移

编码	31 26	25 21	20 16	15 0	
	bgtz 000111	rs	0 00000	offset	
	6	5	5	16	
格式	bgtz rs, offset				
描述	if (GPR[rs] > 0) then 转移				

操作	if (GPR[rs] > 0)
	PC ← PC + 4 + sign_extend(offset‖0^2)
	else
	PC ← PC + 4				

示例	bgtz $s1, -2				
其他					

BGEZ：大于等于 0 时转移

编码	31 26	25 21	20 16	15 0	
	000001	rs	bgez 00001	offset	
	6	5	5	16	
格式	bgez rs, offset				
描述	if (GPR[rs] >= 0) then 转移				

操作	if (GPR[rs] >= 0)
	PC ← PC + 4 + sign_extend(offset‖0^2)
	else
	PC ← PC + 4				

示例	bgez $s1, -2				
其他					

BLTZ：小于 0 时转移

编码	31 26	25 21	20 16	15 0
	000001	rs	bltz 00000	offset
	6	5	5	16
格式	bltz rs, offset			
描述	if (GPR[rs] < 0) then 转移			

操作	if (GPR[rs] < 0)
	PC ← PC + 4 + sign_extend(offset‖0^2)
	else
	PC ← PC + 4			

示例	bltz $s1, -2			
其他				

DIV：符号除

编码	31 26	25 21	20 16	15 6	5 0
	special 000000	rs	rt	0 00 0000 0000	div 011010
	6	5	5	10	6
格式	div rs, rt				
描述	(HI, LO) ← GPR[rs] / GPR[rt] 商存放在LO寄存器，余数存放在HI寄存器				
操作	LO← GPR[rs]div GPR[rt] HI← GPR[rs]mod GPR[rt]				
示例	div $s1, $s2				
其他	如果 GPR[rt]为 0，则 HI/LO 结果不可预料。				

ERET：异常返回

编码	31 26	25 21	20 16	15 11	10 6	5 0
	COP0 010000	80000 1000 0000 0000 0000 0000				eret 011000
	6	20				6
格式	eret					
描述	eret 将保存在 CP0 的 EPC 寄存器中的现场(被中断指令的下一条地址)写入 PC，从而实现从中断、异常或指令执行错误的处理程序中返回。					
操作	PC ← CP0[epc]					

示例	eret
其他	当程序被硬件中断、执行 sc 指令、指令执行异常(如除 0)时，PC 将被保存在 EPC 中。【注意】如果是硬件中断和 SC，EPC 中保存的 PC+4；如果是指令执行异常(如除零、异常等)，则保存 PC。

J：跳转

编码	31 26 25 0		
	j 000010	instr_index	
	6	26	
格式	`j target`		
描述	j 指令是 PC 相关的转移指令。当把 4GB 划分为 16 个 256MB 区域，j 指令可以在当前 PC 所在的 256MB 区域内任意跳转。		
操作	`PC ← PC31..28‖instr_index‖0²`		
示例	`j Loop_End`		
其他	如果需要跳转范围超出了当前 PC 所在的 256MB 区域内时，可以使用 JR 指令。		

JAL：跳转并链接

编码	31 26 25 0	
	jal 000011	instr_index
	6	26
格式	`jal target`	
描述	jal 指令是函数指令，PC 转向被调用函数，同时将当前 PC+4 保存在 GPR[31]中。当把 4GB 划分为 16 个 256MB 区域，jal 指令可以在当前 PC 所在的 256MB 区域内任意跳转。	
操作	`PC ← PC31..28‖instr_index‖0²` `GPR[31] ← PC + 4`	
示例	`jal my_function_name`	
其他	jal 与 jr 配套使用。jal 用于调用函数，jr 用于函数返回。当所调用的函数地址超出了当前 PC 所在的 256MB 区域内时，可以使用 jalr 指令。	

JALR：跳转并链接

编码	31 26	25 21	20 16	15 11	10 6	5 0
	special 000000	rs	0 00000	rd	0 00000	jalr 001001
	6	5	5	5	5	6
格式	`jalr rd, rs`					
描述	jalr 指令是函数指令，PC 转向被调用函数(函数入口地址保存在 GPR[rs]中)，同时将当前 PC+4 保存在 GPR[rd]中。					
操作	`PC ← GPR[rs]` `GPR[rd] ← PC + 4`					
示例	`jalr $s1, $31`					

JR：跳转至寄存器

编码	31 26	25 21	20 11	10 6	5 0
	special 000000	rs	0 00 0000 0000	0 00000	jr 001000
	6	5	10	5	6
格式	`jr rs`				
描述	`PC ← GPR[rs]`				
操作	`PC ← GPR[rs]`				
示例	`jr $31`				
其他	jr 与 jal/jalr 配套使用。jal/jalr 用于调用函数，jr 用于函数返回。				

LB：加载字节

编码	31 26	25 21	20 16	15 0
	lb 100000	base	rt	offset
	6	5	5	16
格式	`lb rt, offset(base)`			
描述	`GPR[rt] ← memory[GPR[base]+offset]`			

操作	`Addr ← GPR[base] + sign_ext(offset)`
	`memword ← memory[Addr]`
	`byte ← Addr`$_{1..0}$
	`GPR[rt] ← sign_ext(memword`$_{7+8*byte..8*byte}$`)`			

示例	lb $v1, 3($s0)			

LUI：立即数加载至高位

编码	31 26	25 21	20 16	15 0
	lui 001111	0 00000	rt	immediate
	6	5	5	16
格式	`lui rt, immediate`			
描述	`GPR[rt] ← immediate‖0`^{16}			
操作	`GPR[rt] ← immediate‖0`^{16}			
示例	`lui $s1, 0x55AA`			
其他				

LW：加载字

编码	31 26	25 21	20 16	15 0
	lw 100011	base	rt	offset
	6	5	5	16
格式	`lw rt, offset(base)`			
描述	`GPR[rt] ← memory[GPR[base]+offset]`			
操作	`Addr ← GPR[base] + sign_ext(offset)`			

	`GPR[rt] ← memory[Addr]`
示例	`lw $v1, 8($s0)`
约束	Addr 必须是 4 的倍数(即 $Addr_{1..0}$ 必须为 00)，否则产生地址错误异常

SLL：逻辑左移

编码	31 26	25 21	20 16	15 11	10 6	5 0
	special 000000	0	rt	rd	s	sll 000000
	6	00000	5	5	5	6
格式	`sll rd, rt, s`					
描述	`GPR[rd] ← GPR[rt] << s`					
操作	`GPR[rd] ←` $GPR[rt]_{(31-s)..0} \| 0^s$					
示例	`sll $s1, $s2, 5`					
其他	`sll $0, $0, 0` 对应的指令码是 `0x0000_0000`，也被认为是 `NOP`(空操作指令)。该指令有时被用于空循环，有时被编译器用于与体系结构相关的编译优化。					

SW：存储字

编码	31 26	25 21	20 16	15 0
	sw 101011	base	rt	offset
	6	5	5	16
格式	`sw rt, offset(base)`			
描述	`memory[GPR[base]+offset] ← GPR[rt]`			
操作	`Addr ← GPR[base] + sign_ext(offset)` `memory[Addr] ← GPR[rt]`			
示例	`sw $v1, 8($s0)`			
约束	Addr 必须是 4 的倍数(即 $Addr_{1..0}$ 必须为 00)，否则产生地址错误异常			