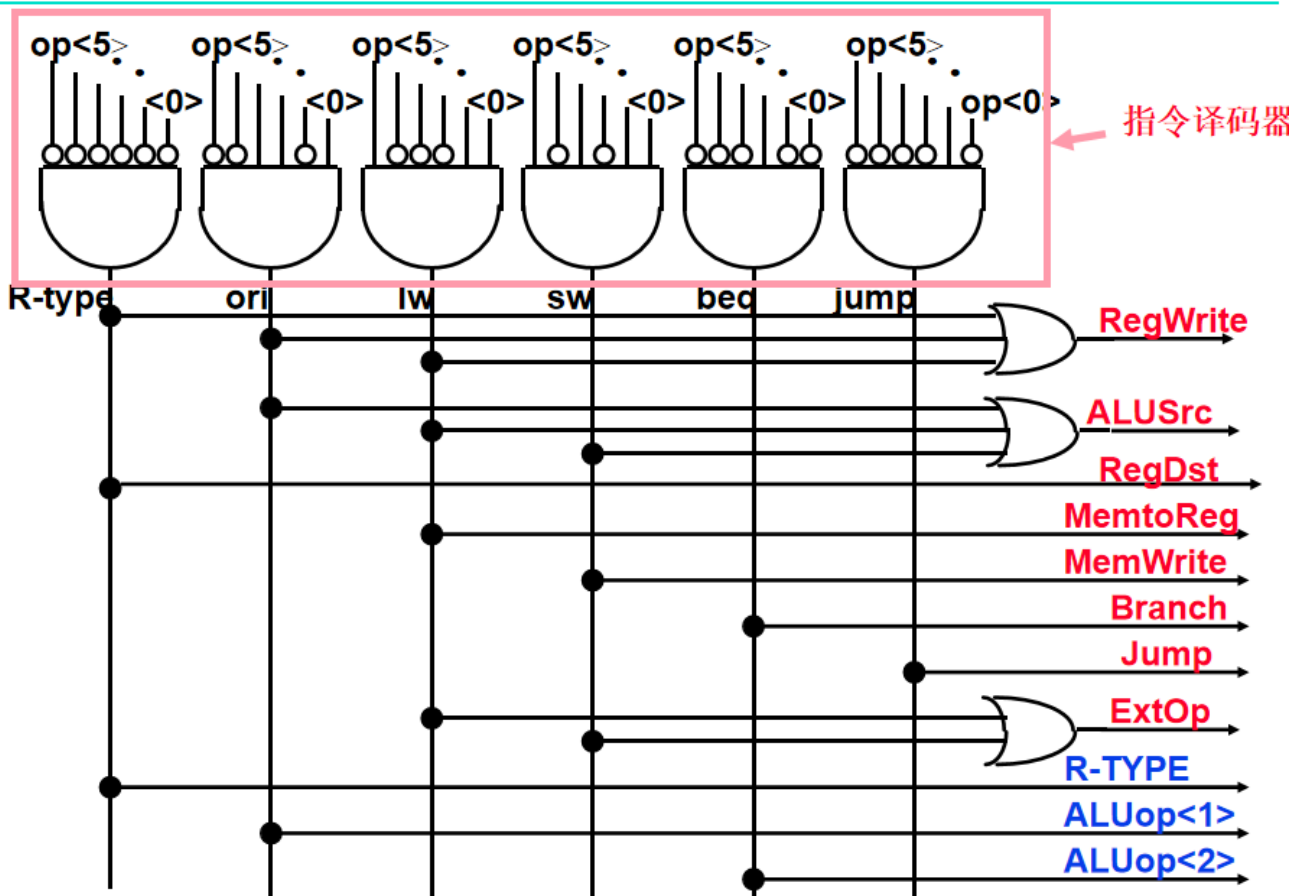
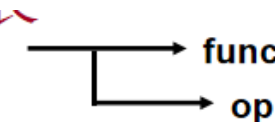


实验四：指令译码器

设计思路



singlelength 67



	func	10 0000	10 0010	We Don't Care :-)				
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
		add	sub	ori	lw	sw	beq	jump
RegDst		1	1	0	0	x	x	x
ALUSrc		0	0	1	1	1	0	x
MemtoReg		0	0	0	1	x	x	x
RegWrite		1	1	1	1	0	0	0
MemWrite		0	0	0	0	1	0	0
Branch		0	0	0	0	0	1	0
Jump		0	0	0	0	0	0	1
ExtOp		x	x	0	1	1	x	x
ALUctr<2:0>		Add	Subtr	Or	Add	Add	Subtr	xxx

1. 模块声明：

- 输入：6位宽的操作码 `op`、6位宽的功能码 `func`。
- 输出：一系列控制信号，包括 `RegWr`、`Branch`、`Jump`、`ExtOp`、`AluSrc`、`MemWr`、`MemtoReg`、`RegDst` 以及 `AluCtrl`。

2. 译码逻辑：

- 使用 `always @(op)` 块，当输入的操作码 `op` 变化时，根据操作码和功能码进行译码，并设置相应的控制信号。
- 使用 `case` 语句进行多路选择，根据操作码的不同，设置相应的控制信号。

3. 操作码分支：

- 对于不同的操作码，根据功能码进一步细分，设置对应的控制信号。

4. 功能码判断：

- 在每个操作码的分支中，根据功能码 `func` 的不同，设置相应的控制信号。

5. 控制信号设置：

- 根据功能码和操作码的判断，设置输出的控制信号。

6. ALU控制信号设置：

- 根据功能码的不同，设置ALU控制信号 `AluCtrl`。

这样，该模块实现了一个简单的MIPS指令译码逻辑，根据输入的操作码和功能码产生相应的控制信号，用于后续的数据通路控制。

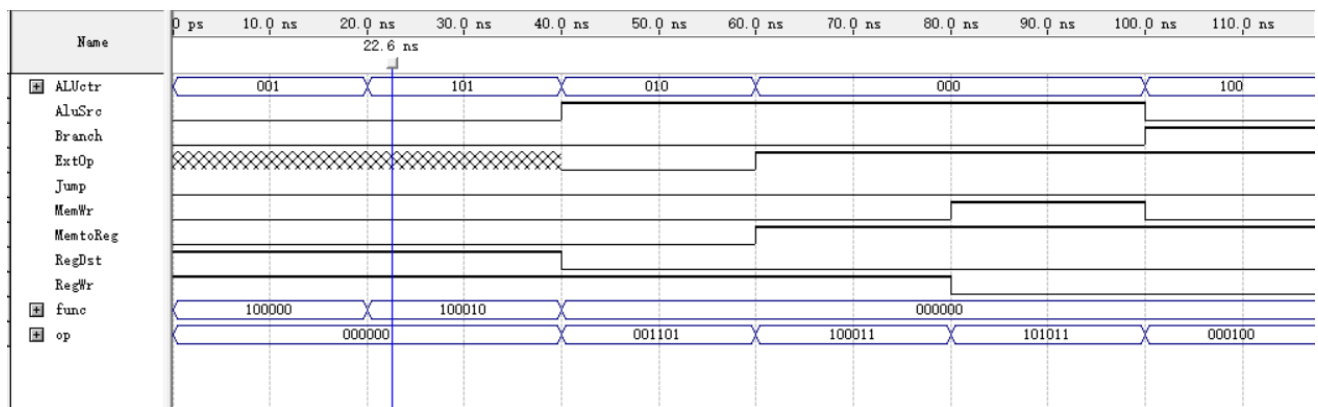
代码展示

```
module
    decoding(op,func,RegWr,Branch,Jump,ExtOp,AluSrc,AluCtrl,MemWr,MemtoReg,RegDst);
input  [5:0] op,func;
output RegWr,Branch,Jump,ExtOp,AluSrc,MemWr,MemtoReg,RegDst;
output [2:0] AluCtrl;
reg RegWr,Branch,Jump,ExtOp,AluSrc,MemWr,MemtoReg,RegDst;
reg [2:0] AluCtrl;

always @(op)
    case(op)
        6'b000000:
            begin
                {Branch,Jump,RegDst,AluSrc,MemtoReg,RegWr,MemWr}=7'b0010010;
                case(func)
                    6'b100000:AluCtrl=3'b001;
                    6'b100010:AluCtrl=3'b101;
                    6'b100011:AluCtrl=3'b100;
                    6'b101010:AluCtrl=3'b111;
```

```
        6'b101011:Aluctr=3'b110;
    endcase
end
6'b001101:
begin
    {Branch,Jump,RegDst,AluSrc,MemtoReg,RegWr,MemWr,ExtOp}=8'b00010100;
    Aluctr=3'b010;
end
6'b001001:
begin
    {Branch,Jump,RegDst,AluSrc,MemtoReg,RegWr,MemWr,ExtOp}=8'b00010101;
    Aluctr=3'b000;
end
6'b100011:
begin
    {Branch,Jump,RegDst,AluSrc,MemtoReg,RegWr,MemWr,ExtOp}=8'b00011101;
    Aluctr=3'b000;
end
6'b101011:
begin
    {Branch,Jump,AluSrc,RegWr,MemWr,ExtOp}=6'b001011;
    Aluctr=3'b000;
end
6'b000100:
begin
    {Branch,Jump,AluSrc,RegWr,MemWr}=5'b10000;
    Aluctr=3'b100;
end
6'b000010:
begin
    {Branch,Jump,RegWr,MemWr}=4'b0100;
end
endcase
endmodule
```

波形分析



当 op=000000，func 分别为 100000，100010 时，为 add，sub 指令，
add:ALUctr=001,AluSrc=0,Branch=0,Jump=0,MemWr=0,MemtoReg=0,RegDst=1,RegWr=1, 译码正确。

sub:ALUctr=101,AluSrc=0,Branch=0,Jump=0,MemWr=0,MemtoReg=0,RegDst=1,RegWr=1, 译码正确。

当op分别为001101,100011,101011,000100时，为ori,lw,sw,beq指令。

ori :
 ALUctr=010,AluSrc=1,Branch=0,ExtOp=0,Jump=0,MemWr=0,MemtoReg=0,RegDst=0,RegWr=1,译码正确。

lw :
 ALUctr=000,AluSrc=1,Branch=0,ExtOp=1,Jump=0,MemWr=0,MemtoReg=1,RegDst=0,RegWr=1,译码正确。

sw :
 ALUctr=000,AluSrc=1,Branch=0,ExtOp=1,Jump=0,MemWr=1,MemtoReg=1,RegDst=0,RegWr=0,译码正确。

beq :
 ALUctr=100,AluSrc=0,Branch=1,ExtOp=1,Jump=0,MemWr=0,MemtoReg=1,RegDst=0,RegWr=0,译码正确。