# 跳转测试：

```mips

#beq jal jr

ori $1,1

ori $2,2

beq $1,$1,br\_1

br\_2:

sw $2,0

br\_1:

sw $2,4

beq $1,$2,br\_2

br\_3:

beq $1,$5,br\_4

ori $5,1

beq $2,$2,br\_3

br\_4:

ori $7,6

jal br\_5

ori $6,6

br\_5:

ori $3,3

beq $6,$7,br\_6

jr $ra

br\_6:

ori $4,4

```

```txt

34210001

34420002

10210001

ac020000

ac020004

1022fffd

10250002

34a50001

1042fffd

34e70006

0c000c0c

34c60006

34630003

10c70001

03e00008

34840004

```

```txt

@00003000: $ 1 <= 00000001

@00003004: $ 2 <= 00000002

@00003010: \*00000004 <= 00000002

@0000301c: $ 5 <= 00000001

@00003024: $ 7 <= 00000006

@00003028: $31 <= 0000302c

@00003030: $ 3 <= 00000003

@0000302c: $ 6 <= 00000006

@00003030: $ 3 <= 00000003

@0000303c: $ 4 <= 00000004

```

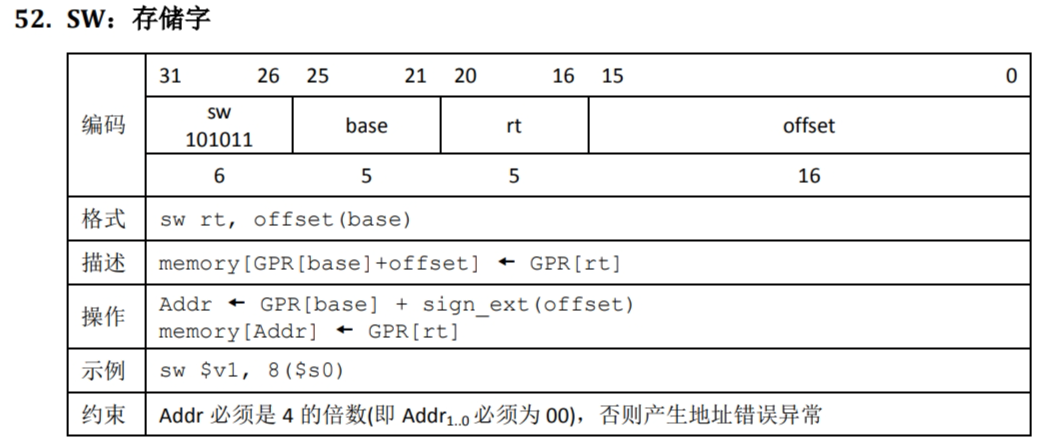
# 思考题：

1. 阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即 32bit × 1024字），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0] ？



addr信号来源于ALU模块计算，为4的倍数。

DM按字寻址，addr11：2即为存储空间按字长划分后的序号。



1. 思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。

**指令对应的控制信号如何取值(清晰但冗长)：**

case (Op)

    `R\_type: begin

        Branch = 1'b0;

        RegWriteByC = 1'b1;

jal = 1'b0;

        …

    end

    `lw: begin

        Branch = 1'b0;

        RegWriteByC = 1'b1;

        WriteAofReg = 2'b00;

        …

end

…

endcase

**控制信号每种取值所对应的指令（简洁）：**

assign MemToReg = lw | add | sub| …;

assign MemWrite = sw;

1. 在相应的部件中，复位信号的设计都是**同步复位**，这与 P3 中的设计要求不同。请对比**同步复位**与**异步复位**这两种方式的 reset 信号与 clk 信号优先级的关系。

同步复位：clk信号>reset信号 （时钟上升沿是复位的前提）

异步复位：reset信号>clk信号 （只要复位信号到来就复位）

1. C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分。

在忽略溢出的前提下，addi 与 addiu 是等价的，因为两者都是将立即数加到寄存器中的值上，并将结果存储到目标寄存器中。由于忽略溢出，无论是有符号数还是无符号数，计算结果都不会影响最终结果。add与addu等价同理。