阶段一之实验五译码器讲义

[阶段一之实验五译码器讲义 1](#_Toc150169591)

[1、 实验目的、重点、难点 1](#_Toc150169592)

[2、 实验任务 1](#_Toc150169593)

[2.1 译码整体模块设计 1](#_Toc150169594)

[2.2 通过指令类型分析译码模块中所包含的译码器的种类和个数 3](#_Toc150169595)

[2.2.1 MIPS指令种类 3](#_Toc150169596)

[2.2.2 指令格式总结 3](#_Toc150169597)

[2.2.3 对操作码部分译码 3](#_Toc150169598)

[2.3 得到运算类型码 4](#_Toc150169599)

[2.4 对操作数进行解析 5](#_Toc150169600)

[2.5 思考 6](#_Toc150169601)

[3、 verilog知识补充 6](#_Toc150169602)

# 

# 阶段一之实验五译码器讲义

## 实验目的、重点、难点

实验目的：

1.了解MIPS指令集中的运算指令，学会对这些指令进行归纳分类。

2.熟悉并掌握译码器的原理、功能和设计。

3.进一步加强运用 verilog 语言进行电路设计的能力。

4.为后续设计 cpu 的实验打下基础。

实验重点：

译码模块在整个cpu设计中的作用。

实验难点：

译码器的原理、功能和设计。

## 实验任务

### 2.1 译码整体模块设计

学习MIPS指令集，熟知指令类型，了解指令功能和编码，译码器实现框图2.1。译码器输入输出端口说明如表2.1。

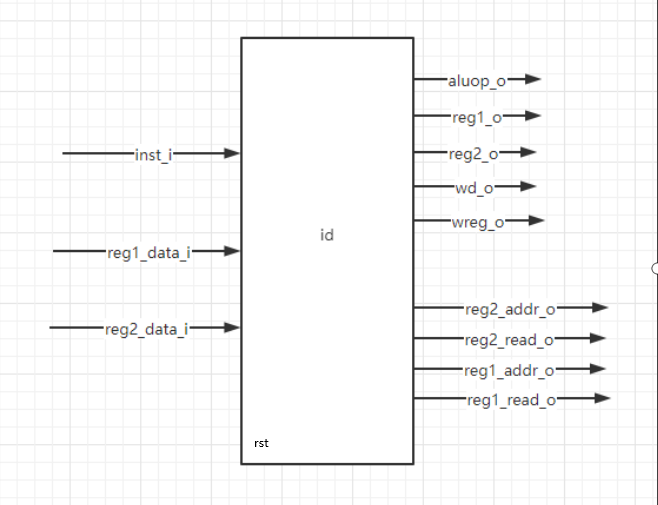


图2.1 译码模块图

表2.1 译码端口说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | inst\_i | 32 | 输入 | 译码阶段的指令 |
| 3 | reg1\_data\_i | 32 | 输入 | 从regfile读入数据1 |
| 4 | reg2\_data\_i | 32 | 输入 | 从regfile读入数据2 |
| 5 | aluop\_o | 14 | 输出 | 译码阶段运算类型 |
| 6 | reg1\_o | 32 | 输出 | 译码阶段源操作数1 |
| 7 | reg2\_o | 32 | 输出 | 译码阶段源操作数2 |
| 8 | wd\_o | 5 | 输出 | 目的寄存器地址，inst\_i[15:11] |
| 9 | wreg\_o | 1 | 输出 | 是否要写入目的寄存器 |
| 10 | reg2\_addr\_o | 5 | 输出 | 第二个寄存器地址，inst\_i[20:16] |
| 11 | reg2\_read\_o | 1 | 输出 | regfile第二个寄存器读使能信号 |
| 12 | reg1\_addr\_o | 5 | 输出 | 第一个寄存器地址,inst\_i[25:21] |
| 13 | reg1\_read\_o | 1 | 输出 | regfile第一个寄存器读使能信号 |

### 2.2 通过指令类型分析译码模块中所包含的译码器的种类和个数

#### 2.2.1 MIPS指令种类

MIPS指令主要由I-Type,J-Type,R-Type三种格式组成。如图2.2所示。

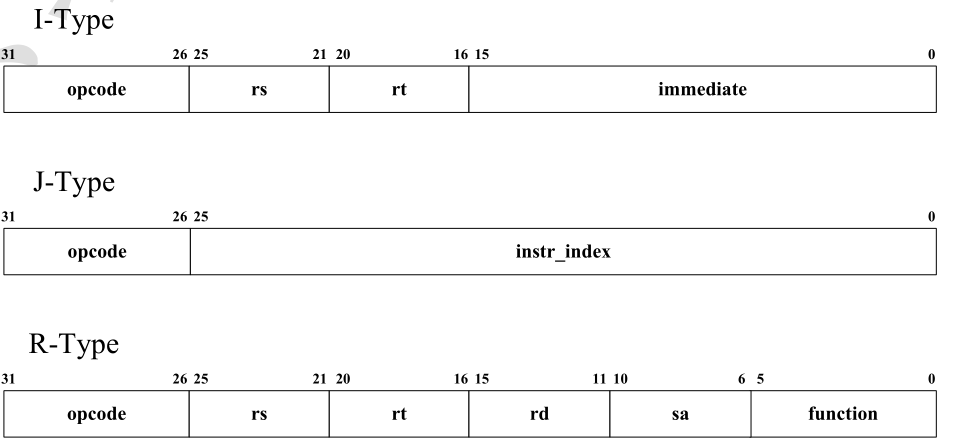


图2.2 MIPS指令种类

#### 2.2.2 指令格式总结

我们的实验任务主要针对R-Type和I-Type设计CPU.综合R-Type和I-Type的格式，我们得到我们要进行的指令格式如图2.3所示。

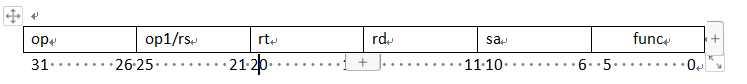


图2.3 指令格式总结

如图2.3所示，我们需要将指令进行划分，对操作码部分进行译码，从而区分出不同的指令;对操作数部分进行截取分析，从而得到操作数的地址或者是操作数本身。

#### 2.2.3 对操作码部分译码

经过上述分析，可以得知用来译码的操作码部分为4段，分别是op（6bit）,op1(5bit),sa(5bit),func(6bit)。由此可分析得知我们需要2个6-64译码器，以及2个5-32译码器。如表2.2所示，为4个译码器的输入和输出。

表2.2 所需4个译码器信号说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 作用 |
| 1 | op | 6 | 对应inst\_i[31:26],用于译码输入 |
| 2 | op\_d | 64 | op译码输出 |
| 3 | op1 | 5 | 对应inst\_i[25:21],用于译码输入 |
| 4 | op1\_d | 32 | op1译码输出 |
| 5 | sa | 5 | 对应inst\_i[10:6],用于译码输入 |
| 6 | sa\_d | 32 | sa译码输出 |
| 7 | func | 6 | 对应inst\_i[5:0] |
| 8 | func\_d | 64 | func译码输出 |

根据4个译码器的输出，我们可以得到相应的指令。如表2.3所示。这样根据4个译码器的输出就可以得到相应的指令。

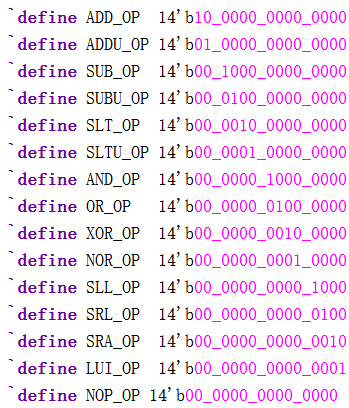
表2.3 根据译码器输出得到相应指令

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 指令信号 | Op\_d | Sa\_d | Op1\_d | Func\_d |
| 1 | inst\_add | OpZero | SaZero |  | FuncAdd |
| 2 | inst\_addu | OpZero | SaZero |  | FuncAddu |
| 3 | inst\_sub | OpZero | SaZero |  | FuncSub |
| 4 | inst\_subu | OpZero | SaZero |  | FuncSubu |
| 5 | inst\_slt | OpZero | SaZero |  | FuncSlt |
| 6 | inst\_sltu | OpZero | SaZero |  | FuncSltu |
| 7 | inst\_and | OpZero | SaZero |  | FuncAnd |
| 8 | inst\_or | OpZero | SaZero |  | FuncOr |
| 9 | inst\_xor | OpZero | SaZero |  | FuncXor |
| 10 | inst\_nor | OpZero | SaZero |  | FuncNor |
| 11 | inst\_sll | OpZero |  | Op1Zero | FuncSll |
| 12 | inst\_srl | OpZero |  | Op1Zero | FuncSrl |
| 13 | inst\_sra | OpZero |  | Op1Zero | FuncSra |
| 14 | inst\_lui | OpLui |  | Op1Zero |  |

### 2.3 得到运算类型码

根据译码出的相应信号，进行编码，得到指令类型，为运算器做准备。我们根据表2.3得到，我们分别用1bit的信号去标识不同的指令，这样可以得到14bit的编码，这个编码同一时刻，只会有1bit的信号为1，这样的编码我们叫做独热码，我们可以直接拿来做为运算类型码。如代码2.1所示。

代码2.1 运算编码独热码标识



思考：这个14位的编码，也可以做为编码器的输入，编码器的输出作为运算码，那编码器的输出应该是几位？

### 2.4 对操作数进行解析

除了操作码部分，我们还需对操作数进行分析。需要分析出送到运算器模块的两个源操作，以及目的操作数。通过分析，我们得知译码器需要译码出寄存器读使能reg1\_read\_o和reg2\_read\_o,以及写使能wreg\_o信号；读地址reg1\_addr\_o, reg2\_addr\_o以及写地址wd\_o。读写使能和指令的关系，如表2.4所示。读写地址和指令的关系如表2.5所示。

表2.4 读写使能和指令的关系

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 使能\指令 | add | addu | sub | subu | slt | sltu | and | or | xor | nor | sll | srl | sra | lui |
| reg1\_read\_o | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| reg2\_read\_o | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| wreg\_o | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| aluop\_o | 独热码编码 | | | | | | | | | | | | | |

表2.5 读写地址说明

|  |  |
| --- | --- |
| 寄存器 | 说明 |
| reg1\_addr\_o | inst\_i[25:21] |
| reg2\_addr\_o | inst\_i[20:16] |
| wd\_o | inst\_lui?inst\_i[20:16]:inst\_i[15:11] |

源操作数的来源除了寄存器堆，还来自于立即数，通过分析可以得知，这14条指令中，比较特殊的有sll,sra,srl这三条移位指令，还有lui指令。sll,sra,srl这三条移位指令源操作数1的来源位立即数sa。Lui指令源操作数1和源操作数2的来源是inst\_i[15:0]。

### 2.5 思考

将本节课的内容与取值模块，以及寄存器堆连接起来，应该怎么连接，最后怎么测试。

## verilog知识补充

（1）generate 语句有主要三种结构：

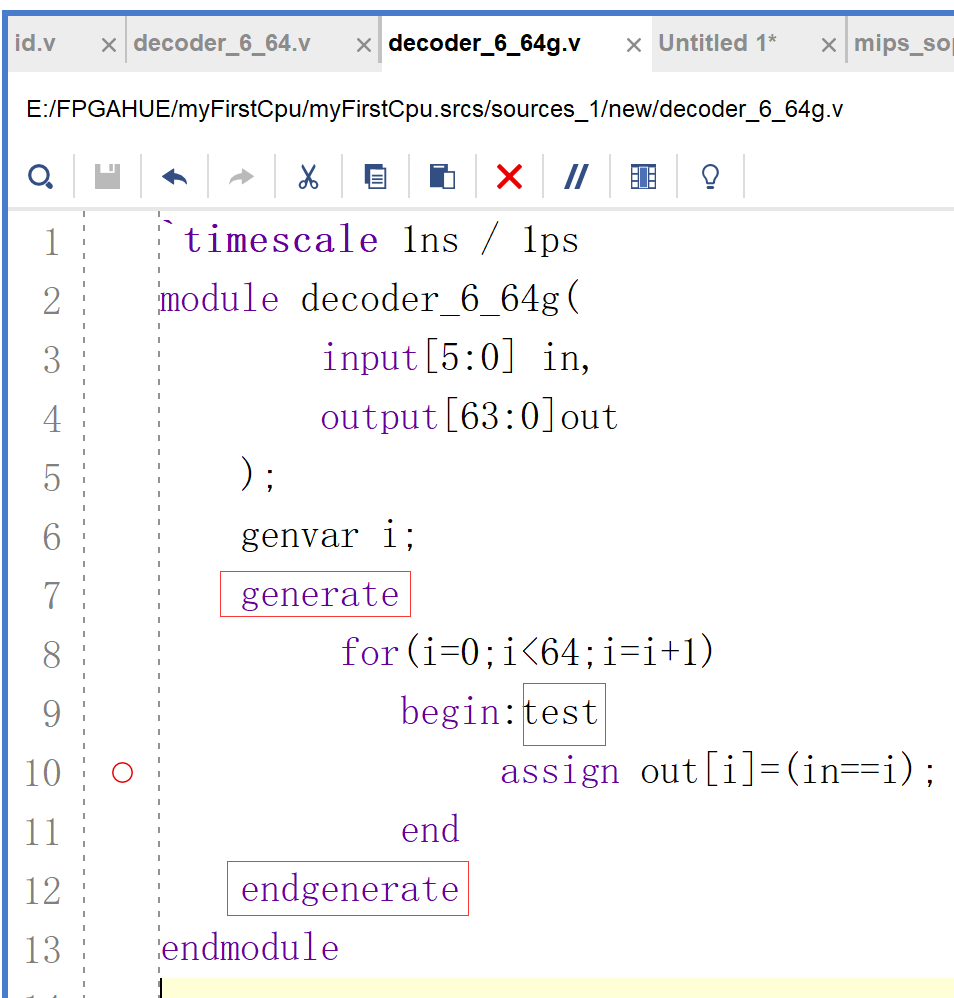
generate - for 语句结构

generate - if 语句结构

generate - case 语句结构

(2)我们主要使用generate-for语句，对另外两种结构不做学习。generate – for, 可以用于编写结构相同但参数不同的赋值语句。如以下示例代码所示。

在使用前必须先声明一个 genvar 变量，用于 for 循环判断，一个 genvar 变量可用于多个 generate 循环。但使用同一个 genvar 变量的 generate 语句不能相互嵌套。genvar 只有在建模时会出现，仿真时不会出现这个变量。在 generate - for 循环结构中，for循环中必须使用begin，end。begin必须要有名称，也就是必须要有标签，例如以下代码中的“test”。



使用generate生成的代码，相当于以下内容。

`timescale 1ns / 1ps

module decoder\_6\_64(

input[5:0] in,

output[63:0]out

);

assign out[0]=(in==6'd0);

assign out[1]=(in==6'd1);

assign out[2]=(in==6'd2);

assign out[3]=(in==6'd3);

assign out[4]=(in==6'd4);

assign out[5]=(in==6'd5);

assign out[6]=(in==6'd6);

assign out[7]=(in==6'd7);

assign out[8]=(in==6'd8);

assign out[9]=(in==6'd9);

assign out[10]=(in==6'd10);

assign out[11]=(in==6'd11);

assign out[12]=(in==6'd12);

assign out[13]=(in==6'd13);

assign out[14]=(in==6'd14);

assign out[15]=(in==6'd15);

assign out[16]=(in==6'd16);

assign out[17]=(in==6'd17);

assign out[18]=(in==6'd18);

assign out[19]=(in==6'd19);

assign out[20]=(in==6'd20);

assign out[21]=(in==6'd21);

assign out[22]=(in==6'd22);

assign out[23]=(in==6'd23);

assign out[24]=(in==6'd24);

assign out[25]=(in==6'd25);

assign out[26]=(in==6'd26);

assign out[27]=(in==6'd27);

assign out[28]=(in==6'd28);

assign out[29]=(in==6'd29);

assign out[30]=(in==6'd30);

assign out[31]=(in==6'd31);

assign out[32]=(in==6'd32);

assign out[33]=(in==6'd33);

assign out[34]=(in==6'd34);

assign out[35]=(in==6'd35);

assign out[36]=(in==6'd36);

assign out[37]=(in==6'd37);

assign out[38]=(in==6'd38);

assign out[39]=(in==6'd39);

assign out[40]=(in==6'd40);

assign out[41]=(in==6'd41);

assign out[42]=(in==6'd42);

assign out[43]=(in==6'd43);

assign out[44]=(in==6'd44);

assign out[45]=(in==6'd45);

assign out[46]=(in==6'd46);

assign out[47]=(in==6'd47);

assign out[48]=(in==6'd48);

assign out[49]=(in==6'd49);

assign out[50]=(in==6'd50);

assign out[51]=(in==6'd51);

assign out[52]=(in==6'd52);

assign out[53]=(in==6'd53);

assign out[54]=(in==6'd54);

assign out[55]=(in==6'd55);

assign out[56]=(in==6'd56);

assign out[57]=(in==6'd57);

assign out[58]=(in==6'd58);

assign out[59]=(in==6'd59);

assign out[60]=(in==6'd60);

assign out[61]=(in==6'd61);

assign out[62]=(in==6'd62);

assign out[63]=(in==6'd63);

endmodule