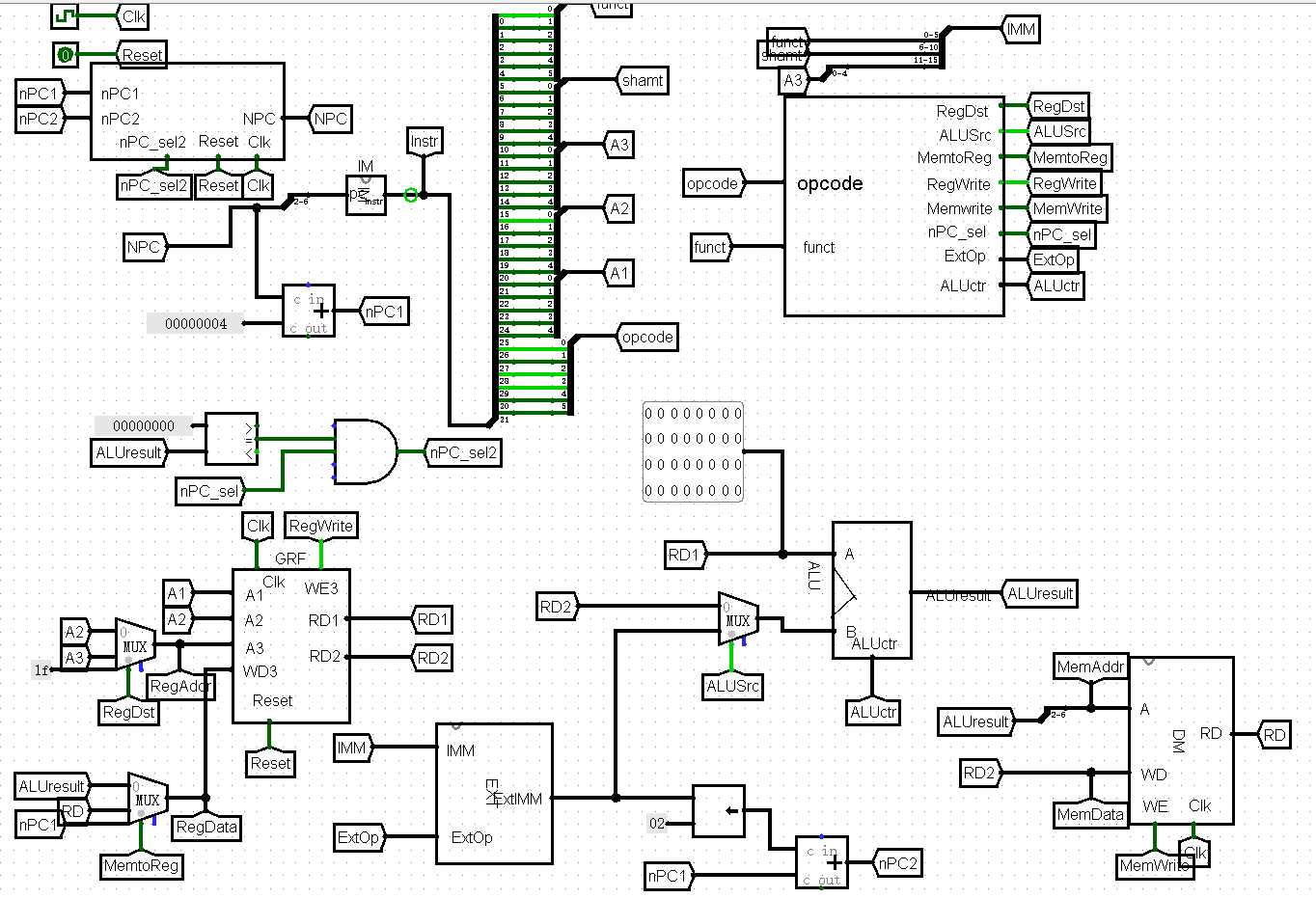
P4实验设计报告

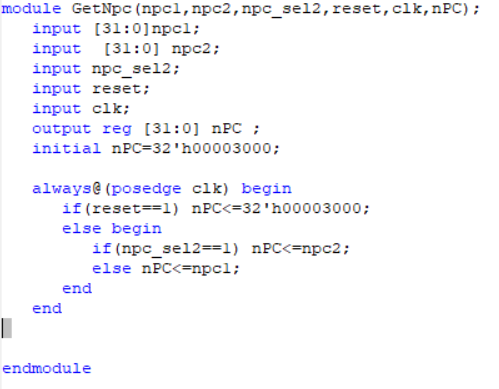
18373085 张海渝

1. 数据通路 
2. 模块规格
3. GetNpc：（程序计数器）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | npc1[31:0] | I | 为PC+4 |
| 2 | npc2[31:0] | I | 解释见下 |
| 3 | npc\_sel2[31:0] | I | 选择下一PC值 |
| 4 | reset | I | 同步复位 |
| 5 | clk | I | 时钟信号 |
| 6 | nPC[31:0] | O | 下一PC值 |

模块解释：

1. 通过选择器进行先选择下一PC值。
2. 如果Reset信号有效时，用选择器将PC值复位。
3. npc2指的是在jal,jr,beq中选择出来的那个pc值。



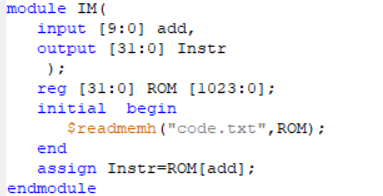
图一：GetNpc模块代码图

1. IM：（指令获取）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | add[9:0] | I | 指令地址 |
| 2 | Instr | O | 输出的机器码 |

模块解释：

1. add[11:2]是当前PC值的第6-2位(因为ROM地址为5位，并且PC每次加4，相当于ROM加1)。
2. 用$readmemh载入机器码。
3. 用寄存器储存机器码。



图二：IM模块代码图

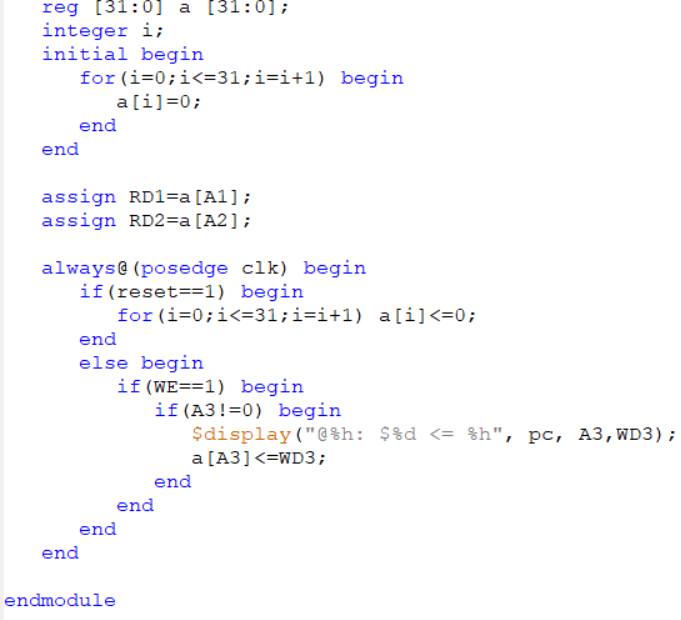
1. GRF：（通用寄存器组，也称寄存器文件、寄存器堆）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | pc[31] | I | 当前PC值 |
| 2 | A1[4:0] | I | 机器码的第25-21位，寄存器编号 |
| 3 | A2[4:0] | I | 机器码的第20-16位，寄存器编号 |
| 4 | A3[4:0] | I | 需要改变的寄存器 |
| 5 | WD3[31:0] | I | 寄存器需要改变为的值 |
| 6 | reset | I | 同步复位信号 |
| 7 | clk | I | 时钟信号 |
| 8 | WE | I | 是否写入寄存器信号 |
| 9 | RD1[31:0] | O | A1寄存器中的值 |
| 10 | RD2[31:0] | O | A2寄存器中的值 |

模块解释：

1.0号寄存器始终为0，不改变。

2.一共有32个带有使能端的寄存器。

3.每次寄存器改变时，输

图三：GRF模块部分代码

4.ALU：（算数逻辑单元）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | A[31:0] | I | 进行操作的第一个数 |
| 2 | B[31:0] | I | 进行操作的第二个数 |
| 3 | ALUctr[2:0] | I | 进行操作的方式 |
| 4 | ALUresult[31:0] | O | 计算后的结果 |

模块解释：

1. ALUctr=0：输出0值

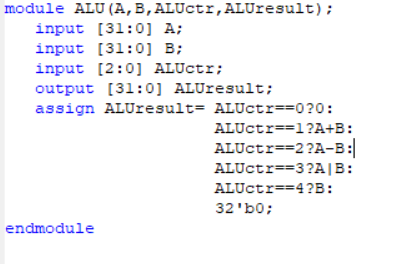
ALUctr=1：加法

ALUctr=2：减法

ALUctr=3：或（or）操作

ALUctr=4:输出B的值

ALUctr=其他：输出0



图四：ALU模块代码图

1. DM：（数据存储器）

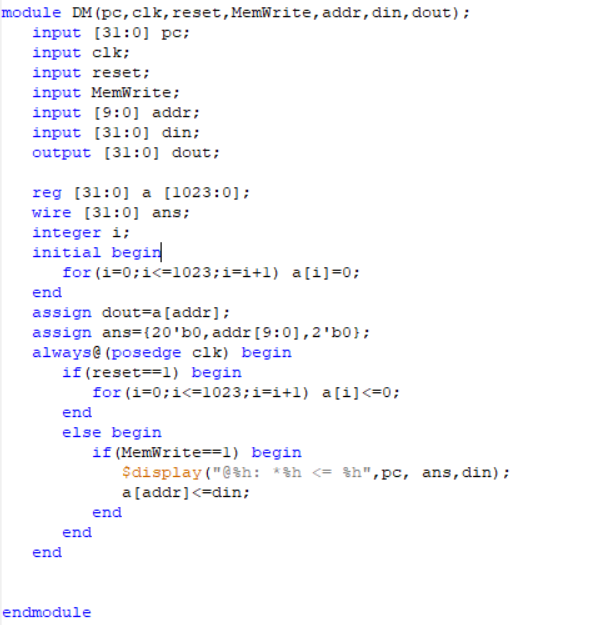
|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | addr[9:0] | I | 进行操作的数据的地址 |
| 2 | din[31:0] | I | 数据 |
| 3 | WE | I | 控制是否RAM是否工作 |
| 4 | Clk | I | 时钟信号 |
| 5 | Dout[31:0] | O | 读出的数据 |
| 6 | pc | I | 当前PC值 |

模块解释：

1.此模块通过一组寄存器。

2.因为模块容量为4kb，所以进行操作的地址为ALU计算结果的11-2位。

3.用ans代表当前addr的32位表达值。



图五：DM模块代码图

6.EXT：（数据扩展器）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | IMM[15:0] | I | 进行扩展的数 |
| 2 | ExtOp[1:0] | I | 扩展的方式 |
| 3 | ExtIMM[31:0] | O | 扩展后的数据 |

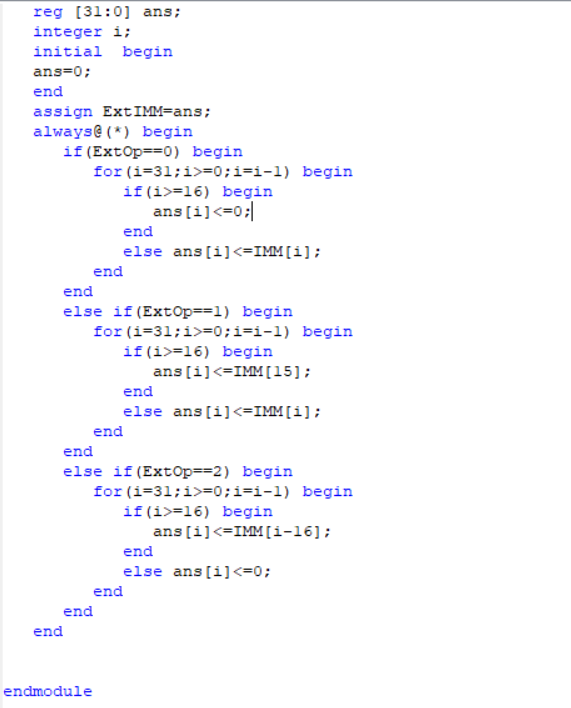
模块解释：

1. ExtOp=0：前16位0扩展

ExtOp=1：符号扩展至32位

ExtOp=2：后16位补0扩展

2，用ans（一个寄存器）以及for循环的利用实现。



图六：EXT模块代码图

7．Control：（控制器）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能名称 | 方向 | 功能描述 |
| 1 | opcode[5:0] | I | Special |
| 2 | funct[5:0] | I | Function |
| 3 | RegDst[1:0] | O | 选择需要在GRF中写入数据的寄存器 |
| 4 | ALUSrc | O | 选择进行ALU操作的第二个操作数 |
| 5 | MemtoReg[1:0] | O | 选择存入GRF中寄存器的数据 |
| 6 | RegWrite | O | 是否在GRF中写入控制信号 |
| 7 | MemWrite | O | DM中的RAM使能端控制信号 |
| 8 | nPC\_sel | O | 选择下一PC值信号 |
| 9 | ExtOp[1:0] | O | 数据扩展方式 |
| 10 | ALUctr[2:0] | O | ALU进行操作的方式 |
| 11 | npc\_sel3 | O | 解释见下 |
| 12 | npc\_sel4 | O | 解释见下 |

模块解释：

1. 首先通过opcode与funct选择出此时进行操作的指令方式。

2.通过if\_else语句实现控制信号的选择。

3. RegDst==2时选择31号寄存器，MemtoReg==2是选择PC-4，为了实现jal指令。

4.npc\_sel3表示在beq和（jal，jr）之中选择是哪一种指令。

5.npc\_sel4表示在jal和jr之中选择是哪一种指令。

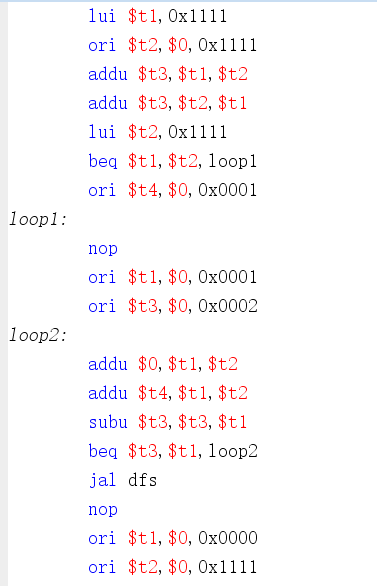
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| funct | 100000 | 100010 |  |  |  |  |  |
| opcode | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
|  | add | sub | ori | lw | sw | beq | lui |
| RegDst[1:0] | 1 | 1 | 0 | 0 | x | x | 0 |
| ALUScr | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| Memto [1:0] | 0 | 0 | 0 | 1 | x | x | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| nPC\_sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ExtOp[1:0] | x | x | 0 | 1 | 1 | x | 2 |
| ALUctr[2:0] | 1 | 2 | 3 | 1 | 1 | 2 | 4 |
| npc\_sel3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| npc\_sel4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

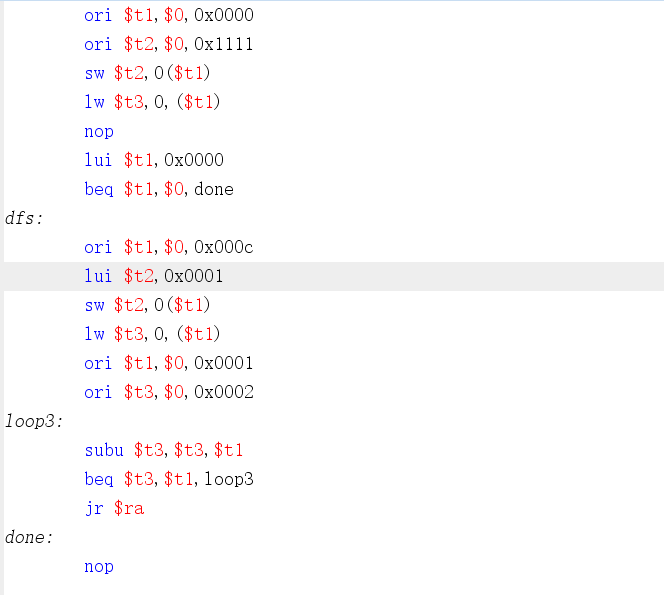
图七：信号输出真值表（Memto代表MemtoReg）

|  |  |  |  |
| --- | --- | --- | --- |
| funct | 001000 | 000000 |  |
| opcode | 000000 | 000000 | 000011 |
|  | jr | nop | jal |
| RegDst[1:0] | 0 | 0 | 2 |
| ALUScr | 0 | 0 | 0 |
| Memto [1:0] | 0 | 0 | 2 |
| RegWrite | 0 | 0 | 1 |
| MemWrite | 0 | 0 | 0 |
| nPC\_sel | 1 | 0 | 1 |
| ExtOp[1:0] | 0 | 0 | 0 |
| ALUctr[2:0] | 0 | 0 | 0 |
| npc\_sel3 | 1 | 0 | 1 |
| npc\_sel4 | 1 | 0 | 0 |

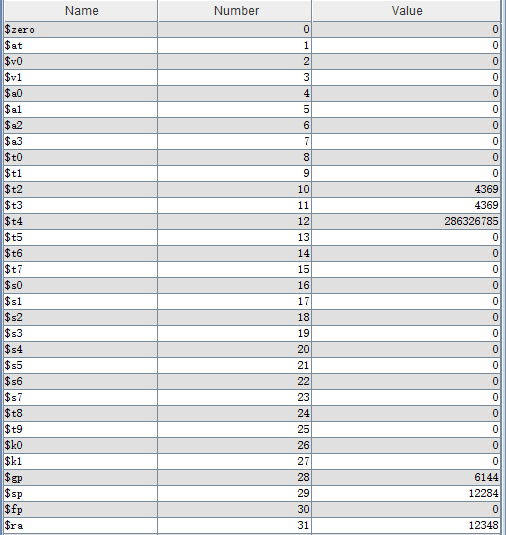
图七（续）：信号输出真值表（Memto代表MemtoReg）

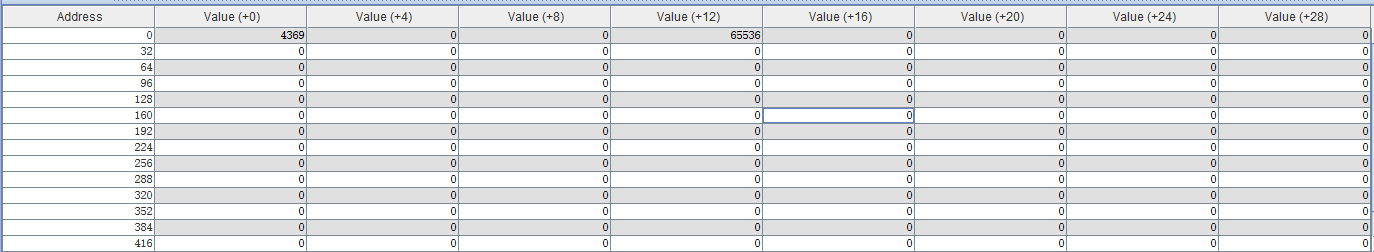
1. 测试代码MIPS测试：代码：



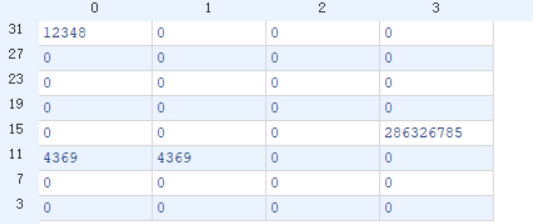
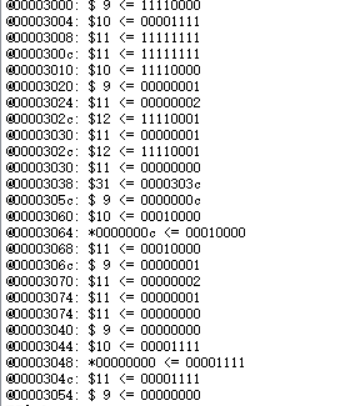


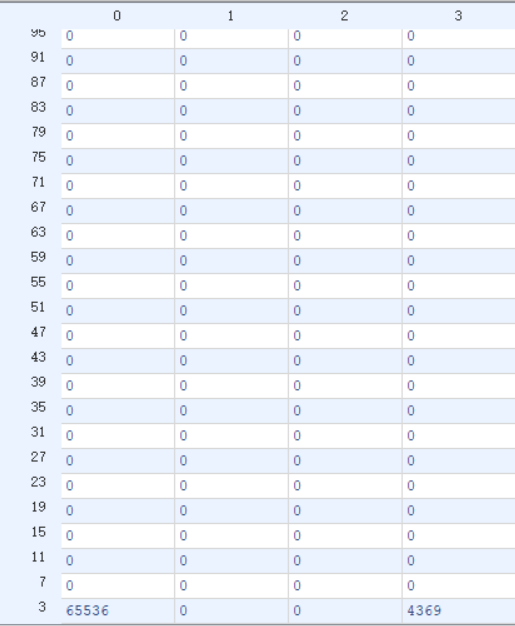
运行结果：（其中的$gp,$sp不进行比较）





ISE运行结果：





1. 思考题

(LO,T2)

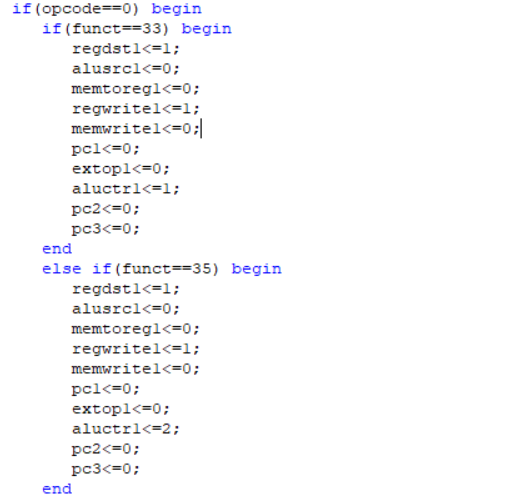
1.地址每次增加4相当于DM中寄存器地址增加1，所以从ALUresult第二位取，又因为DM大小为4KB，所以一共取10位，所以来自于ALUresult，又因为如果直接于这个模块相连，可以取[11:2]位，所以是[11:2],而不是[9:0]。而且来自于ALUresult。

2.对PC值，GRF中的寄存器，以及DM中的数据储存器，因为在仿真过程中，如果不将这些置为初始值，那么在运算过程中会影响计算结果，导致结果出错。

（LO,T3）:

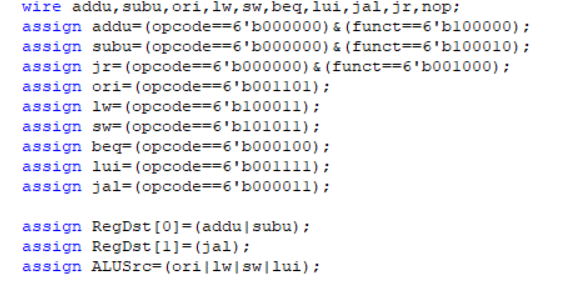
1.

1.应用if\_else语句：

示例：

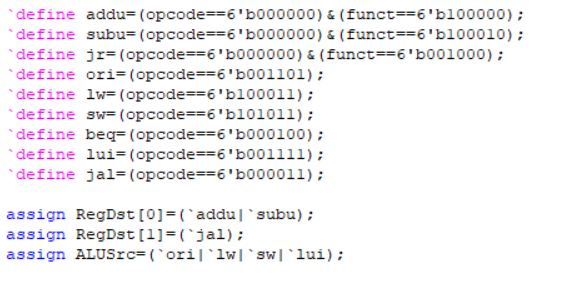
2.assign语句完成操作码和控制信号的值之间的对应。

示例：



3.利用宏定义：

示例：



2. if\_else语句对于我自己来说用的最熟练，我觉得最不容易出错，但是有一个致命的缺点，过于复杂，容易出错。重复的代码语句多。

利用assign语句，有很强的拓展性，利于修改，方便观看，可以很快速的发现问题所在。

宏定义：与assign语句类似，方便快捷，逻辑更为简单。

（LO,T5）:

1.

其中addi指令：GPR[rt]<-GPR[rs]+ immediate

temp <- (GPR[rs]31<-GPR[rs]) + sign\_extend(immediate)

if temp32 ≠ temp31 then SignalException(IntegerOverflow)

else GPR[rt] ← temp31..0

endif

addiu：GPR[rt] <-GPR[rs] + sign\_extend(immediate)

在不考虑溢出的时候，可知addi直接执行else语句，所以等价。

add: temp <- (GPR[rs]31GPR[rs]) + (GPR[rt]31GPR[rt])

if temp32 ≠ temp31 then SignalException(IntegerOverflow)

else GPR[rd] ← temp31..0

endif

addu: GPR[rd] ←GPR[rs] + GPR[rt]

可知不考虑溢出的时候add也是直接执行else语句，所以等价。

2.

优点：不容易发生冲突，一个周期执行一个指令，不会像多周期或者流水线CPU那样会出现各种各样的问题，单周期CPU的搭建简单一些，并且逻辑也较为简单。

缺点：一个周期只能执行一条指令，满足不了现在的大多数功能，运行较慢，最为致命的一个缺点就是速度效率低。

3.

关系：因为在运用jal指令的时候，通常是进入一个函数，而大多数函数都会递归，就是会重复的调用自己，这个时候就需要将当前执行的这一次函数的数据压入栈中，以免运行的时候会出现错误的信息，当调用jr指令回来的时候，通过栈中的值恢复当前需要的一些值，这样才能顺利的进行这一次函数的调用，否则很容易出错。