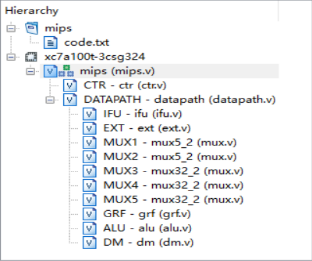
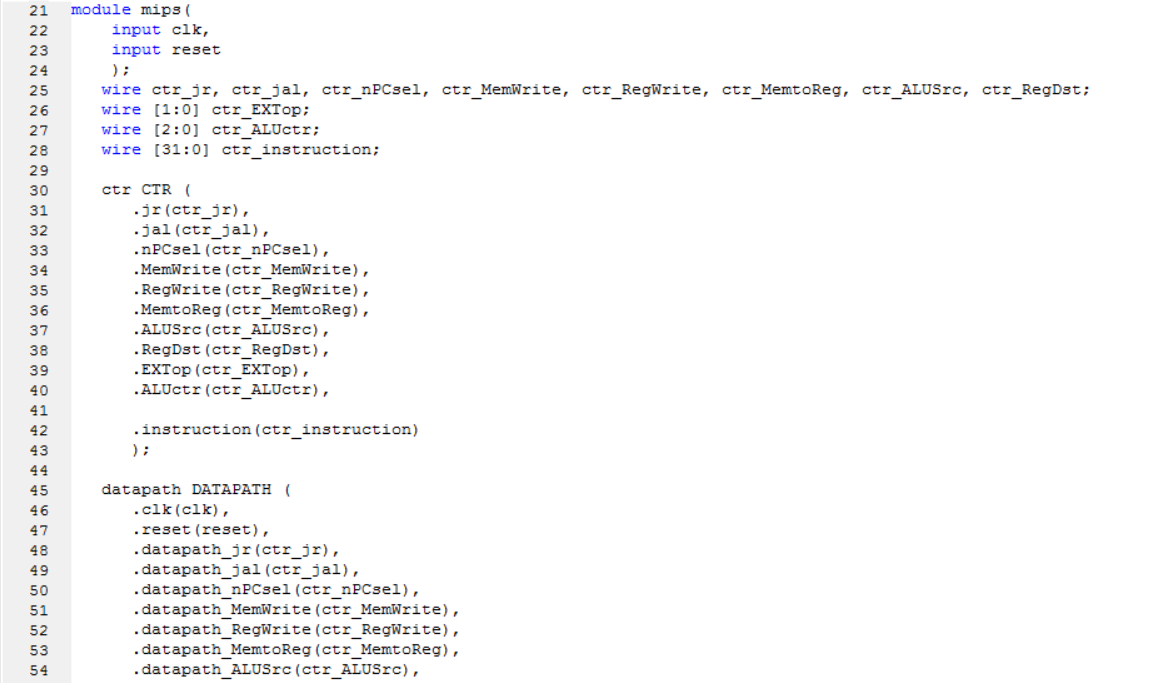
计算机组成原理实验报告

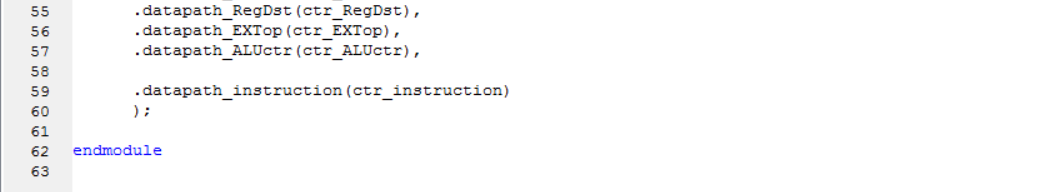
1. CPU设计文档
2. 总体设计



图表 1模块设计

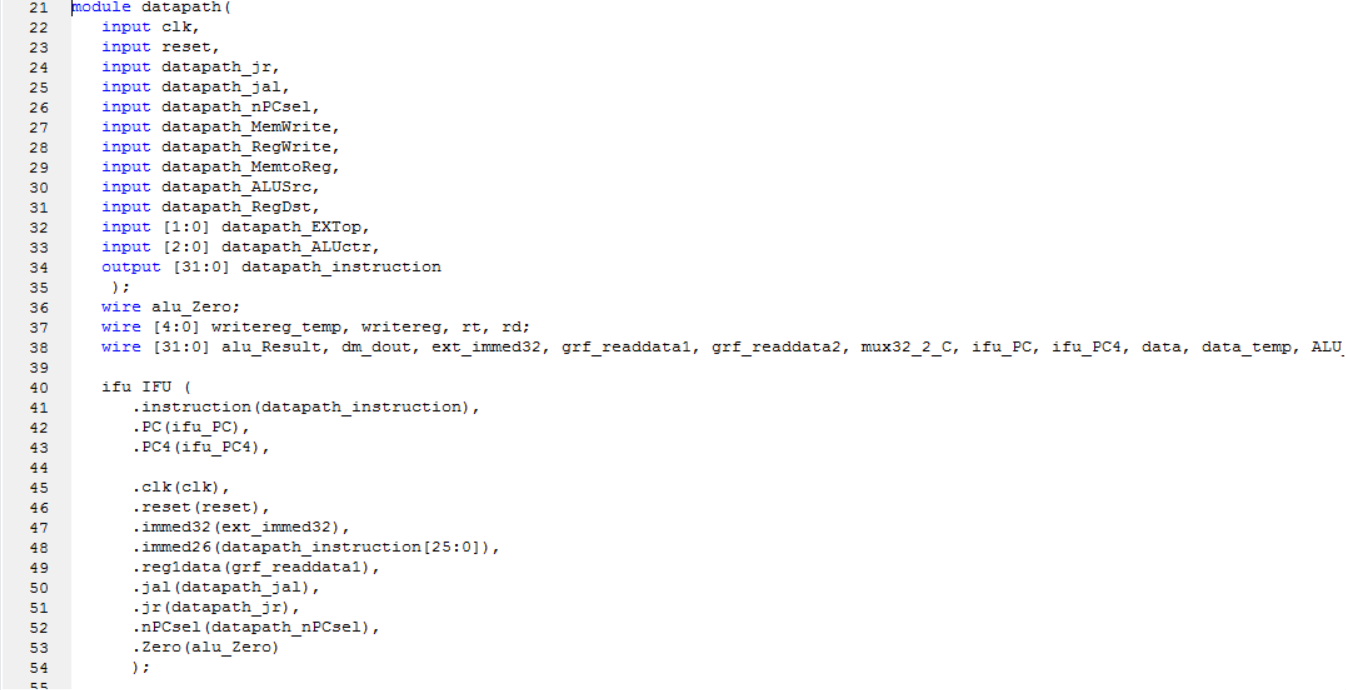


图表 2总体设计

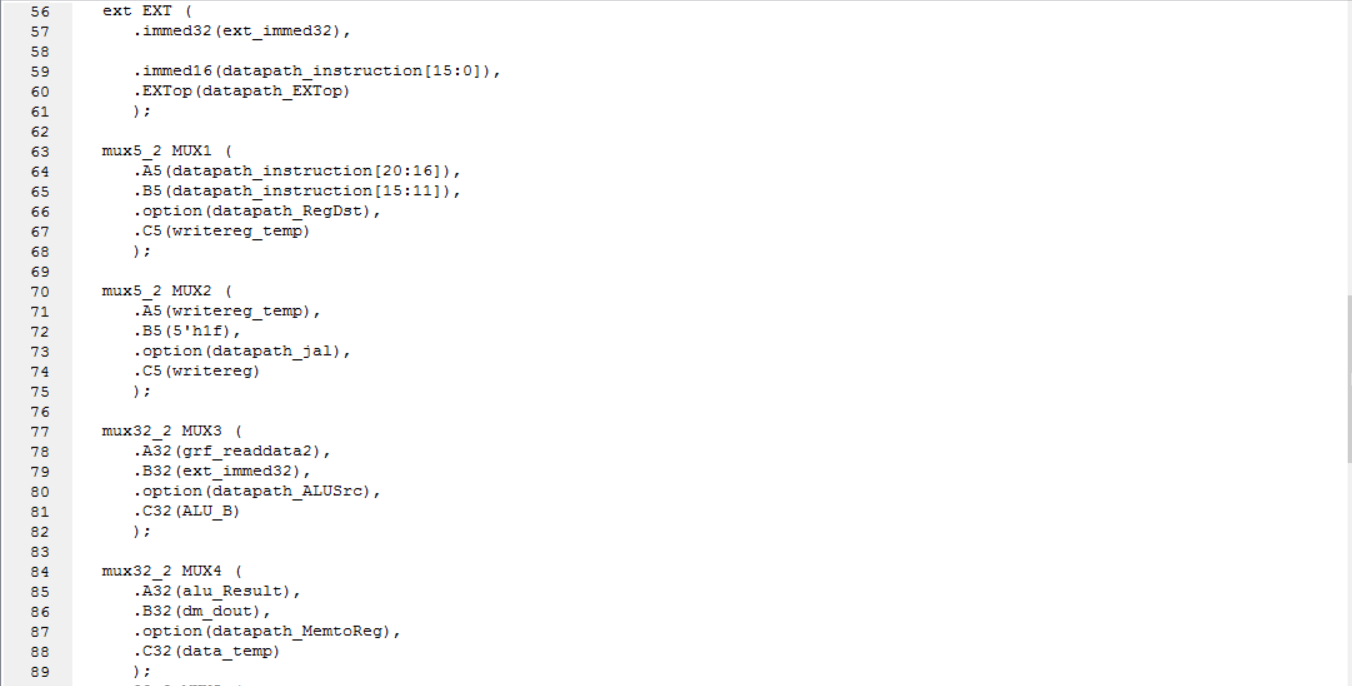


图表 3 总体设计

1. 数据通路设计
2. datapath（数据通路）



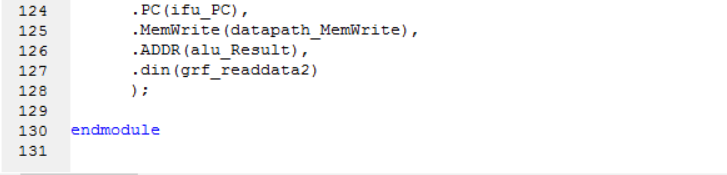
图表 4datapath



图表 5datapath



图表 6datapath



图表 7datapath

1. 端口说明

表格 1datapath端口说明

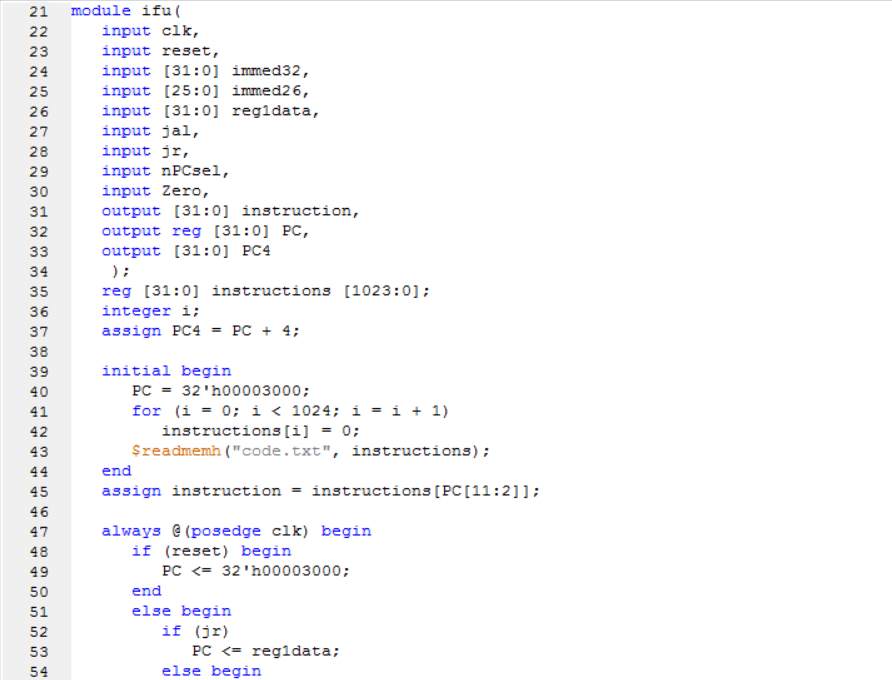
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位  0：无效 |
| datapath\_jr | I | 控制器发来jr信号  1：jr指令  0：无效 |
| datapath\_jal | I | 控制器发来jal信号  1：jal指令  0：无效 |
| datapath\_nPCsel | I | 控制器发来nPCsel信号  1：跳转指令  0：无效 |
| datapath\_MemWrite | I | 控制器发来MemWrite信号  1：写内存  0：无效 |
| datapath\_RegWrite | I | 控制器发来RegWrite信号  1：写寄存器  0：无效 |
| datapath\_MemtoReg | I | 控制器发来MemtoReg信号  1：内存输出有效内容  0：无效 |
| datapath\_ALUSrc | I | 控制器发来ALUSrc信号  1：32位立即数有效  0：无效 |
| datapath\_RegDst | I | 控制器发来RegDst信号  1：写入rd寄存器  0：写入rt寄存器 |
| datapath\_EXTop[1:0] | I | 控制器发来EXTop信号  00：无符号扩展  01：低16位补0  10：有符号扩展  11：有符号扩展后逻辑左移两位 |
| datapath\_ALUctr[2:0] | I | 控制器发来ALUctr信号  控制信号  000：加运算  001：减运算  010：或运算  011：输出写入数据2  100:异或运算 |
| datapath\_instruction[31:0] | O | 发给控制器的指令 |

1. 功能定义

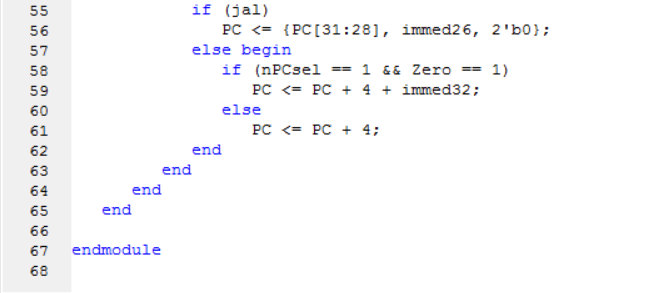
表格 2datapath功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 连接基本模块 | 通过datapath，以声明中间变量和实例化引用的方式连接各基础模块 |

1. ifu（取指令单元）



图表 8ifu



图表 9ifu

1. 端口说明

表格 3ifu端口说明

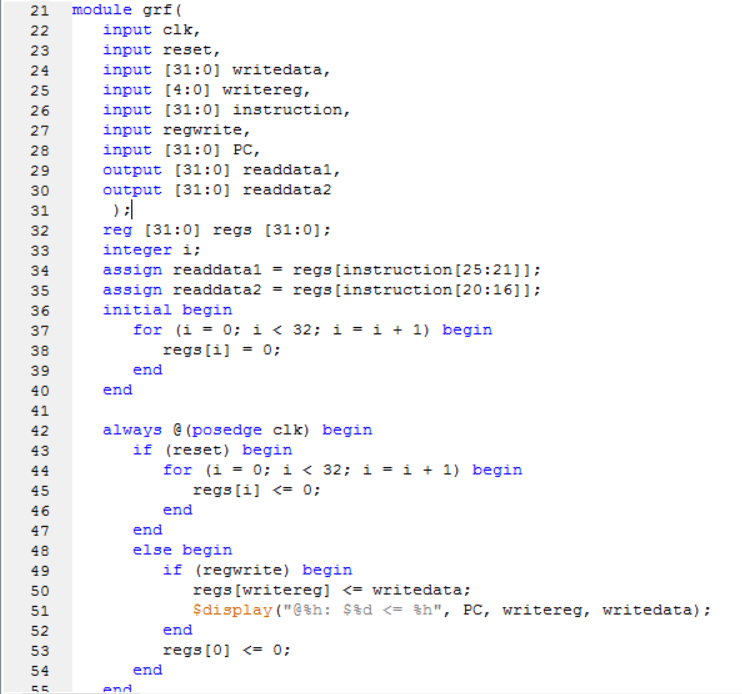
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| jr | I | 控制器发来jr信号  1：jr指令  0：无效 |
| reset | I | 复位信号  1：复位  0：无效 |
| clk | I | 时钟信号 |
| jal | I | 控制器发来jal信号  1：jal指令  0：无效 |
| nPCsel | I | 控制器发来nPCsel信号  1：跳转指令  0：无效 |
| Zero | I | alu发来Zero信号  1：alu两输入相等  0：alu两输入不等 |
| immed32[31:0] | I | ext发来32位立即数信号 |
| immed26[25:0] | I | 26位立即数信号 |
| reg1data[31:0] | I | grf发来reg1的值 |
| instruction[31:0] | O | 输出的指令 |
| PC[31:0] | O | 输出的当前PC |
| PC4[31:0] | O | 输出的当前PC+4 |

1. 功能定义

表格 4 ifu功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，PC被置为0x00000000 |
| 2 | 取指令 | 根据PC从IM中取出指令 |
| 3 | 计算下一条指令地址 | PCPC+4 || PCreg1data || PCPC + 4 + immed32 || PC{PC[31:28], immed26, 2'b0} |

1. grf（通用寄存器组）



图表 10grf

1. 端口说明

表格 5grf端口说明

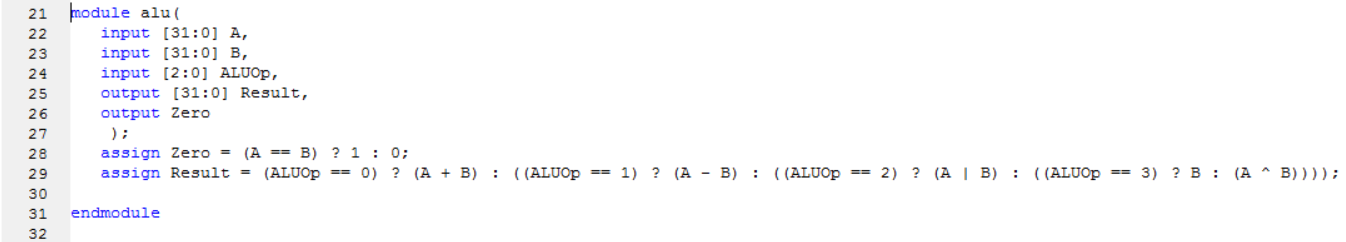
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| writedata[31:0] | I | 写入的32位数据 |
| reset | I | 复位信号  1：复位  0：无效 |
| clk | I | 时钟信号 |
| instruction[31:0] | I | 指令 |
| writereg[4:0] | I | 写寄存器编号 |
| regwrite | I | 写控制信号  1：写入  0：无效 |
| readdata1[31:0] | O | 32位寄存器1输出 |
| readdata2[31:0] | O | 32位寄存器2输出 |

1. 功能定义

表格 6grf功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，32个寄存器被置为0x00000000 |
| 2 | 写寄存器 | 写寄存器控制信号有效时，把32位数据写入寄存器 |
| 3 | 读寄存器 | 根据输入的地址读出两个寄存器中的值 |

1. alu（算术逻辑单元）



图表 11ALU

1. 端口说明

表格 7alu端口说明

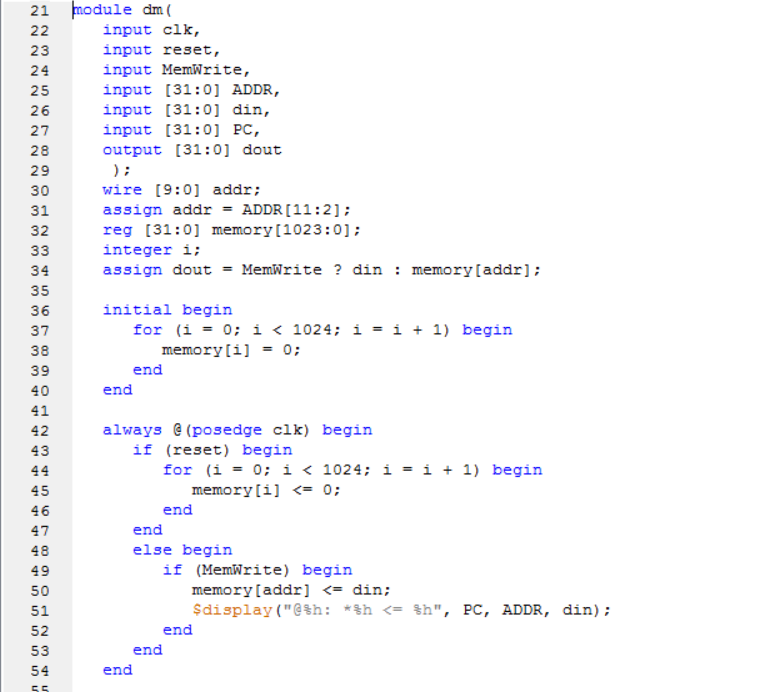
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 32位写入数据1 |
| B[31:0] | I | 32位写入数据2 |
| ALUOp[2:0] | I | 控制信号  000：加运算  001：减运算  010：或运算  011：输出写入数据2  100:异或运算 |
| Zero | O | 相等信号  1：相等  0：不相等 |
| Result[31:0] | O | 32位输出数据 |

1. 功能定义

表格 8alu功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | A+B |
| 2 | 减运算 | A-B |
| 3 | 或运算 | A|B |
| 4 | 输出写入数据2 | B |
| 5 | 异或运算 | A^B |
| 6 | 比较运算 | A==B |

1. dm（数据存储器）



图表 12dm

1. 端口说明

表格 9dm端口说明

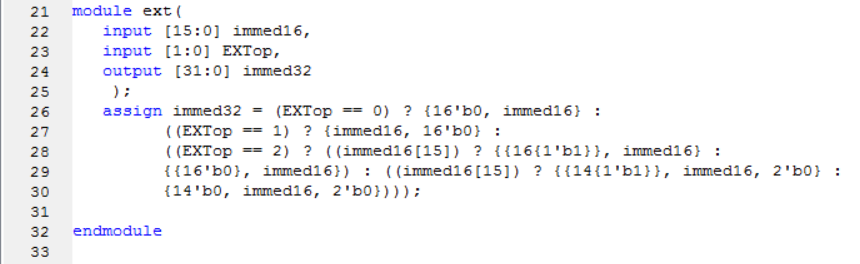
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ADDR [31:0] | I | 32位写入内存地址 |
| din[31:0] | I | 32位写入数据 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位  0：无效 |
| PC[31:0] | I | 当前PC |
| MemWrite | I | 写内存控制信号  1：写入  0：无效 |
| dout[31:0] | O | 32位输出数据 |

1. 功能定义

表格 10dm功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，内存和读出内存的寄存器被置为0x00000000 |
| 2 | 写内存 | 写内存控制信号有效时，根据输入的地址写入32位数据 |
| 3 | 读内存 | 根据输入的地址读出内存数据 |

1. ext（位扩展器）



图表 13ext

1. 端口说明

表格 11ext端口说明

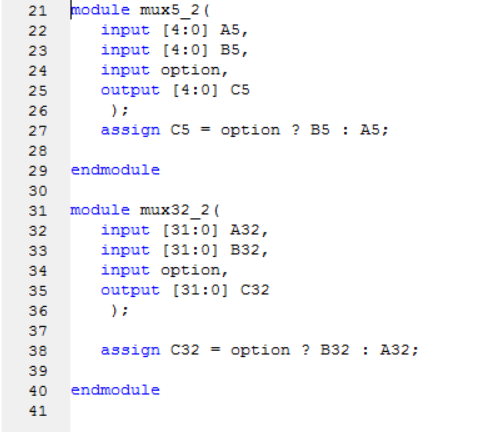
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| immed16[15:0] | I | 16位写入立即数 |
| EXTop[1:0] | I | 扩展控制信号  00：无符号扩展  01：低16位补0  10：有符号扩展  11：有符号扩展后逻辑左移两位 |
| Immed32[31:0] | O | 32位输出立即数 |

1. 功能定义

表格 12ext功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号扩展 | 高16位补0 |
| 2 | 低16位补0 | 低16位补0 |
| 3 | 有符号扩展 | Immed[15]为1时高16位补1，为0时高16位补0 |
| 4 | 有符号扩展后逻辑左移两位 | Immed[15]为1时高16位补1，为0时高16位补0，再左移两位，溢出舍去，低2位补0 |

1. mux（多路选择器）



图表 14mux

1. 端口说明

表格 13mux端口说明

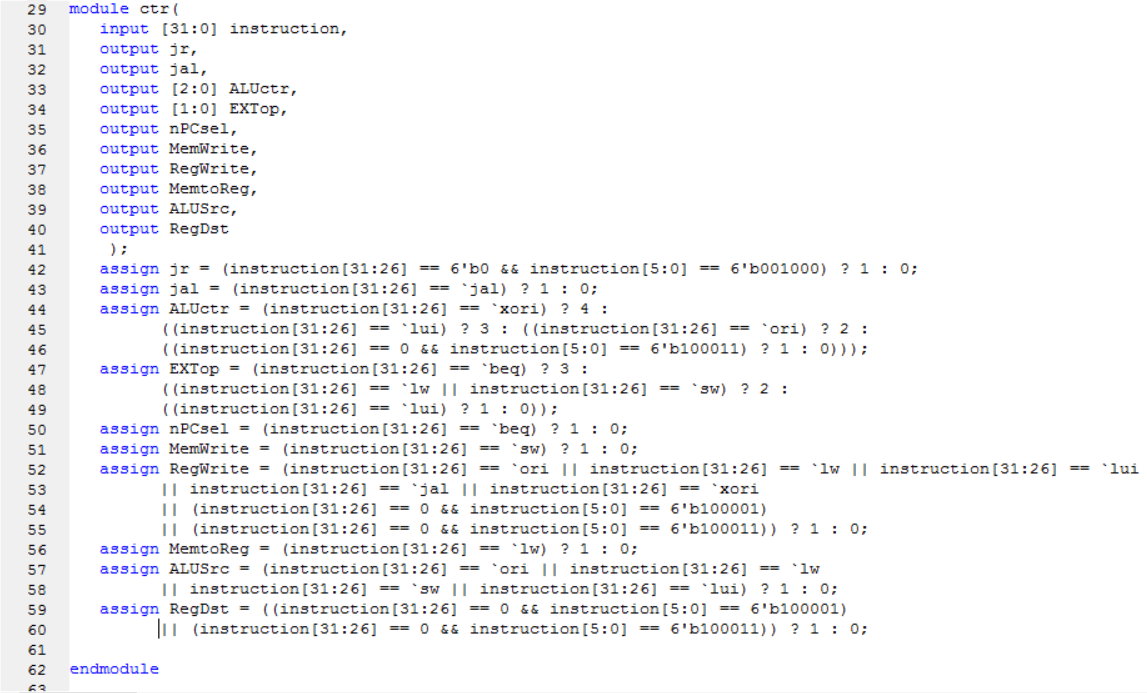
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A5[4:0] | I | 5位输入A5 |
| B5[4:0] | I | 5位输入B5 |
| option | I | 选择控制信号  1：输出B5  0：输出A5 |
| C5[4:0] | O | 5位输出C5 |
| A32[31:0] | I | 32位输入A32 |
| B32[31:0] | I | 32位输入B32 |
| option | I | 选择控制信号  1：输出B32  0：输出A32 |
| C32[31:0] | O | 32位输出C32 |

1. 功能定义

表格 14mux功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 5位输入2选1 | option为1输出B5，为0输出A5 |
| 2 | 32位输入2选1 | option为1输出B32，为0输出A32 |

1. 控制器设计



图表 15ctr

1. 端口说明

表格 15ctr端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| instruction[31:0] | I | 32位指令 |
| RegDst | O | grf写寄存器决定信号  1：rd  0：rt |
| ALUSrc | O | alu输入数据B决定信号  1：32位立即数  0：GRF寄存器2输出值 |
| MemtoReg | O | grf写入数据决定信号  1：DM输出数据  0：ALU输出数据 |
| RegWrite | O | grf写寄存器信号  1：写寄存器  0：无效 |
| MemWrite | O | 写内存dm信号  1：写入内存  0：无效 |
| nPC\_sel | O | 跳转信号  Zero为1时：  1：跳转  0：无效  Zero为0时：  无效 |
| EXTop[1:0] | O | 扩展控制信号  00：无符号扩展  01：低16位补0  10：有符号扩展  11：有符号扩展后逻辑左移两位 |
| ALUctr[2:0] | O | alu控制信号  000：加运算  001：减运算  010：或运算  011：输出写入数据2  100:异或运算 |
| jr | O | jr信号  1：jr指令  0：无效 |
| jal | O | jal信号  1：jal指令  0：无效 |

1. 真值表

表格 16ctr真值表

|  |  |  |  |
| --- | --- | --- | --- |
| func | 100001 | 100011 | 001000 |
| op | 000000 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 | 001110 | 000011 |
|  | addu | subu | jr | ori | lw | sw | beq | lui | nop | xori | jal |
| RegDst | 1 | 1 | 0 | 0 | 0 | x | x | 0 | 0 | 0 | 0 |
| ALUSrc | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| MemtoReg | 0 | 0 | 0 | 0 | 1 | x | x | x | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| nPC\_sel | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| EXTop | x | x | 00 | 00 | 10 | 10 | 11 | 01 | 00 | 00 | 00 |
| ALUctr | 000 | 001 | 000 | 010 | 000 | 000 | x | 011 | 000 | 100 | 000 |
| jr | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| jal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1. 测试程序
2. 测试程序

.data

.text

#测试ori指令

#第三个立即数是无符号扩展，不存在负数的情况

ori $a0, $0, 123 #测试与0进行or运算

ori $a1, $a0, 456 #测试两个非0数的or运算

#测试lui指令

lui $a2, 123 #测试，构造正数

lui $a3, 0xffff #测试，构造负数

#测试addu指令

#无符号相加，不存在负数情况

addu $s0, $a0, $a2 #测试正数相加

#测试subu指令

#无符号相减，不存在负数情况

subu $s1, $a2, $a0 #测试正数相减

#测试sw指令

ori $t0, $t0, 0x0000 #构造0

sw $a0, 0($t0) #测试0111\_1011存入内存

sw $a1, 4($t0) #测试1\_1111\_1011存入内存

#测试lw指令

lw $s0, 0($t0) #测试内存写寄存器

lw $s1, 4($t0) #测试内存写寄存器

#测试beq指令

ori $a0, $0, 1 #构造1

ori $a1, $0, 2 #构造2

ori $a2, $0, 1 #构造1

beq $a0, $a1, loop1 #测试不跳转

nop

beq $a0, $a2, loop2 #测试跳转

nop

loop1:ori $a0, $0, 0

loop2:ori $a1, $0, 0

#测试jal,jr指令

ori $s2, $2, 1

ori $s3, $2, 2

addu $s4, $s2, $s3

jal loop

nop

jal end

nop

ori $s2, $2, 3

loop: ori $s3, $2, 4

jr $ra

end: nop

1. 机器码

3404007b

348501c8

3c06007b

3c07ffff

00868021

00c48823

35080000

ad040000

ad050004

8d100000

8d110004

34040001

34050002

34060001

10850003

00000000

10860002

00000000

34040000

34050000

34520001

34530002

0253a021

0c000c1c

00000000

0c000c1e

00000000

34520003

34530004

03e00008

00000000

1. 期望输出

$a0 0x00000001

$a2 0x00000001

$a3 0xffff0000

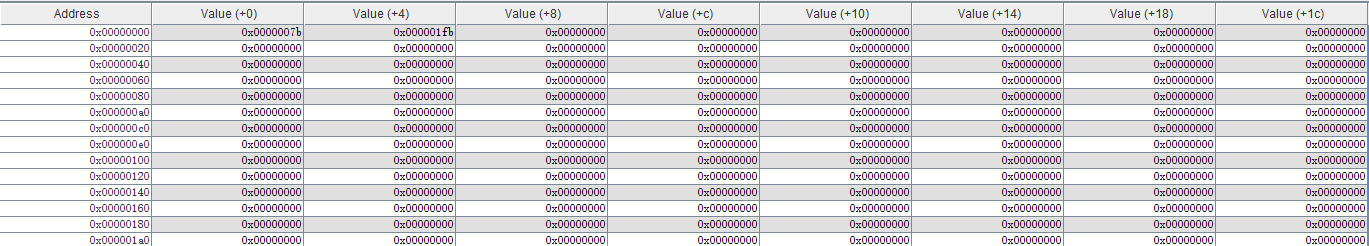
$s0 0x0000007b

$s1 0x000001fb

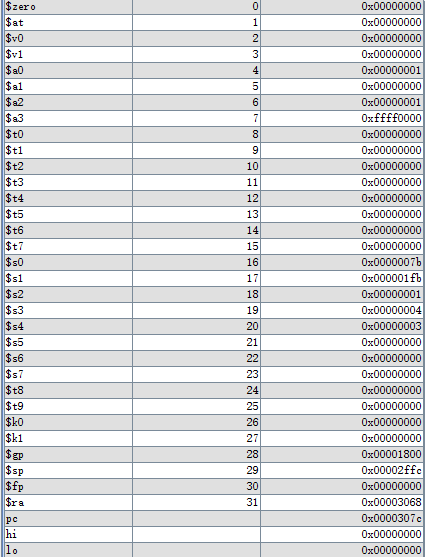
$s2 0x00000001

$s3 0x00000004

$ra 0x00003068



图表 16MARS输出



图表 17MARS输出

1. Verilog输出

@00003000: $ 4 <= 0000007b

@00003004: $ 5 <= 000001fb

@00003008: $ 6 <= 007b0000

@0000300c: $ 7 <= ffff0000

@00003010: $16 <= 007b007b

@00003014: $17 <= 007aff85

@00003018: $ 8 <= 00000000

@0000301c: \*00000000 <= 0000007b

@00003020: \*00000004 <= 000001fb

@00003024: $16 <= 0000007b

@00003028: $17 <= 000001fb

@0000302c: $ 4 <= 00000001

@00003030: $ 5 <= 00000002

@00003034: $ 6 <= 00000001

@0000304c: $ 5 <= 00000000

@00003050: $18 <= 00000001

@00003054: $19 <= 00000002

@00003058: $20 <= 00000003

@0000305c: $31 <= 00003060

@00003070: $19 <= 00000004

@00003064: $31 <= 00003068

1. 结论

期望输出与实际输出相同。

1. 思考题
2. 根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

输入DM的addr位宽为10位， 取得是32位addr的2-11位。直接声明[11:2]addr则能更直观的提醒取得是11：2这10位，实际上与9：0没有区别。从alu输出的result得到。

1. 在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

对ifu、dm、grf进行复位。这些部件有寄存器或ram，清零时需要将这些寄存器或ram清空，来初始化cpu。

1. 列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

利用if-else、case完成操作码和控制信号的值之间的对应，如

case(op)

6’b000011: begin

jal = 1;

……

End

endcase

利用assign语句完成操作码和控制信号的值之间的对应，如

assign jr = (instruction[31:26] == 6'b0 && instruction[5:0] == 6'b001000) ? 1 : 0;

利用宏定义，如

`define beq 6'b000100

assign nPCsel = (instruction[31:26] == `beq) ? 1 : 0;

1. 根据你所列举的编码方式，说明他们的优缺点。

if-else、case方式对每个指令产生控制信号的过程简洁明了，易于添加新指令，但对每个信号的所有产生情况不够清晰；assign对每个信号的所有产生情况集中到一起，但对一个指令所产生的所有信号不清晰，难以debug；宏定义使得上述两种方式更加清晰。

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 。

溢出即计算结果超出32位，对于支持c语言的mips指令，因c语言不处理溢出，addi与addiu的结果相同，都看作没有溢出，即addi无需判断是否溢出并处理overflow。同理add与addu等价，都不处理溢出所得overflow值。

1. 根据自己的设计说明单周期处理器的优缺点。

优点：每一条指令执行过程清晰，直观；

缺点：各个元件功能单一，不能复用，造成芯片面积大、成本高的缺陷。处理n 条指令至少需要n 个周期，周期取决于最长数据通路的耗时，耗时较长。

1. 简要说明jal、jr和堆栈的关系。

jal 跳转至函数，jr 跳转回原程序。函数调用时，$s0-$s7八个寄存器为受保护寄存器，即在函数调用前后，这八个寄存器中的数值不能被修改，若函数对$s0-$s7 有修改，则需要用栈将原来值保存起来。当函数进行递归调用时，程序执行符合栈的先进后出模式，在执行jal 后进栈，jr 返回后出栈，以保证受保护寄存器的值不变。