基于VerilogHDL的单周期CPU设计文档

1. 模块规格
2. IFU

表1 IFU模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | Clk | I | 时钟信号 |
| 2 | PCSrc | I | 选择下一条指令的来源  0:下一条指令的地址为PC+1  1:下一条指令地址为指令中低16位符号扩展后的立即数+PC+1 |
| 3 | Reset | I | 复位PC的信号 0:不复位 1:复位 |
| 4 | JumpAddr[31:0] | I | Jr指令的跳转地址 |
| 5 | instr[31:0] | O | 32位MIPS指令 |
| 6 | JType | O | 当前指令是J型指令时的选择信号 |
| 7 | JorJal | O | 当前指令是J指令或Jal指令时的选择信号 |
| 8 | PCAdd4 | O | 当前PC+4 |
| 9 | PCout  [31:0] | O | 当前PC输出 |

表2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x00000000 |
| 2 | 计算下一条指令的地址 | 若当前指令不是跳转指令，则PC的值为当前PC+4  若当前指令是j型指令，则根据指令进行跳转  若当前指令是跳转指令，则PC的值为当前PC+4+offset |

1. GRF

表3 GRF模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | Clk | I | 时钟信号 |
| 2 | Reset | I | 复位寄存器文件的信号 0:不复位 1:复位 |
| 3 | A1rsBase[4:0] | I | 读寄存器地址1 |
| 4 | A2rt[4:0] | I | 读寄存器地址2/写寄存器地址1 |
| 5 | rd[4:0] | I | 写寄存器地址2 |
| 6 | WriteData[31:0] | I | 写入寄存器的数据 |
| 7 | WE | I | 寄存器写使能 |
| 8 | RegDstA3 | I | 决定写入的寄存器 0:写入A2/rt 1:写入rd |
| 9 | RD1[31:0] | O | 读寄存器的数据1 |
| 10 | RD2[31:0] | O | 读寄存器的数据2 |
| 11 | PC[31:0] | I | 当前PC |

表4 GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有寄存器均被设置为0x00000000 |
| 2 | 读寄存器 | 根据输入的指令和控制单元的信号输出寄存器中的数据 |
| 3 | 写寄存器 | 根据输入的指令和控制单元的信号向应写入的寄存器中写入WriteData中的数据 |

1. ExtU

表5 ExtU模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | ExtOp[1:0] | I | 扩展单元选择信号 |
| 2 | ImmIn[15:0] | I | 输入的16位立即数 |
| 3 | ImmOut[31:0] | O | 扩展后输出的32位立即数 |

表6 ExtU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 零扩展 | ExtOP为00时，将ImmIn零扩展至32位并输出 |
| 2 | 符号扩展 | ExtOP为01时，将ImmIn符号扩展至32位并输出 |
| 3 | 低16位补0 | ExtOP为10时，输出{ImmIn, 16’b0} |

1. ALU

表7 ALU模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | ALUCtrl[2:0] | I | ALU控制信号 |
| 2 | SrcA[31:0] | I | ALU的操作数A |
| 3 | SrcB[31:0] | I | ALU的操作数B |
| 4 | bigger | O | 当ALU做减法时，指示A是否大于B |
| 5 | equal | O | 当ALU做减法时，指示A是否等于B |
| 6 | smaller | O | 当ALU做减法时，指示A是否小于B |
| 7 | ALUResult[31:0] | O | 32位ALU运算结果 |

表8 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法 | ALUCtrl为000时，输出A+B |
| 2 | 减法 | ALUCtrl为001时，输出A-B |
| 3 | 或操作 | ALUCtrl为010时，输出A|B |
| 4 | 有符号比较A和B的大小关系 | 当ALUCtrl为001时，  A>B: A>B信号为1，=信号为0，<信号为0  A=B: A>B信号为0，=信号为1，<信号为0  A<B: A>B信号为0，=信号为0，<信号为1 |

1. DM

表9 DM模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | Clk | I | 时钟信号 |
| 2 | Reset | I | 复位信号 0:不复位 1:复位 |
| 3 | WE | I | 数据存储器写使能 |
| 4 | Addr[31:0] | I | ALU计算出的写数据存储器的地址 |
| 5 | RD[31:0] | O | 32位读出数据 |
| 6 | WD[31:0] | O | 32位写入数据 |
| 7 | PC[31:0] | I | 当前PC |

表10 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，所有数据置零 |
| 2 | 读 | 根据输入的寄存器地址读出数据 |
| 3 | 写 | 根据输入的地址，写入WD输入的数据 |

6.ControlUnit

表11 ControlUnit模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | OPCode[5:0] | I | 输入指令的OPCode |
| 2 | FunctCode[5:0] | I | 输入指令的FunctCode |
| 3 | Men2Reg | O | 写回寄存器的数据流的选择信号 |
| 4 | MenWrite | O | 数据存储器写使能 |
| 5 | Branch[2:0] | O | 判断当前指令是哪条分支指令 |
| 6 | ALUSrc(SrcB) | O | 输入ALU的操作数B的数据流的选择信号 |
| 7 | ALUCtrl[3:0] | O | ALU控制信号 |
| 8 | ExtOp[1:0] | O | 扩展单元控制信号 |
| 9 | RegDstA3[1:0] | O | 写回寄存器的地址的选择信号 |
| 10 | RegWriteEN | O | 寄存器堆写使能 |
| 11 | JType | O | 判断当前指令是否是J型指令 |
| 12 | JorJal | O | 判断当前指令是否是J指令或Jal指令 |

1. 控制器设计

表12 ControlUnit设计

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | ori | lw | sw | beq | lui | nop | jal | jr |
| Op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 | 000011 | 000000 |
| Funct | 100001 | 100011 |  |  |  |  |  |  |  | 001000 |
| PCSrc | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| RegWriteEn | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| RegDst | 1 | 1 | 0 | 0 | x | x | 0 | 0 | 10 | x |
| ExtU | x | x | 00 | 01 | 01 | 01 | 10 | x | x | x |
| ALUSrc(SrcB) | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| ALUCtrl | 0000 | 0001 | 0010 | 0000 | 0000 | 0001 | 0010 | 0000 | x | x |
| Branch | 000 | 000 | 000 | 000 | 000 | 001 | 000 | 000 | 000 | 000 |
| MenWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Men2Reg | 0 | 0 | 0 | 1 | x | 0 | 1 | 0 | 0 | 0 |
| JType | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| JorJal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

1. 测试程序

lui $t0,0xff88

lui $t1 0xfa24

haa:

subu $t6 $t6 $t5

beq $t6 $t5 ha

jr $ra

ha:

subu $t2 $t0 $t1

ori $t3 $t2 0x1b62

beq $t6, $t3, func

sw $t3, 4($0)

lui $t4 0xfa23

nop

beq $t1 $t4 ,ha

lw $t6, 4($0)

lui $t5 0x0001

addu $t4 $t4, $t5

beq $t1 $t4, ha

func:

jal haa

beq $t6, $t3, func

nop

测试期望（运行结果）：

@00003000: $ 8 <= ff880000

@00003004: $ 9 <= fa240000

@00003008: $14 <= 00000000

@00003014: $10 <= 05640000

@00003018: $11 <= 05641b62

@00003020: \*00000004 <= 05641b62

@00003024: $12 <= fa230000

@00003030: $14 <= 05641b62

@00003034: $13 <= 00010000

@00003038: $12 <= fa240000

@00003014: $10 <= 05640000

@00003018: $11 <= 05641b62

@00003040: $31 <= 00003044

@00003008: $14 <= 05631b62

前三次beq指令不进行跳转，第四、第五和第六次beq指令进行跳转，在第五次跳转后通过jal进行函数调用，在第六次beq指令后通过jr指令返回，第七次beq指令不跳转，程序结束运行。

思考题

1. L0.T2
2. 地址是[11:2]是因为检索数据存储器中的数据时是以四的倍数进行检索，这样定义省去了需要右移的麻烦；addr信号来自ALU的计算结果。
3. 把PC恢复到0x0000\_3000，寄存器文件中所有寄存器的值置0，数据存储器中的所有数据置0。这些地方能存储程序运行中的数据，复位时应把这些信号全部置0。
4. L0.T3
5. 利用if-else（或case）完成操作码和控制信号的值之间的对应：

always @\* begin

case (OPCode) begin

6’b000000:begin

case(FuncCode)

6’b000001: begin

RegWriteEN = 1;

ALUCtrl = 0;

......

End

6’b000010: ........

end

6’b000011: begin

......

end

endcase

end

优点：逻辑明晰，方便查找某条特定指令需要的各种信号；

缺点：指令数多的时候代码太长，不方便查找特定指令对应的位置。

1. 利用assign语句完成操作码和控制信号的值之间的对应；

assign R = (OPCode == 6'b000000);

assign addu = (FuncCode == 6'b100001); //R-Type

assign subu = (FuncCode == 6'b100011); //R-Type

assign ori = (OPCode == 6'b001101);

assign lw = (OPCode == 6'b100011);

assign sw = (OPCode == 6'b101011);

assign beq = (OPCode == 6'b000100);

assign lui = (OPCode == 6'b001111);

assign jal = (OPCode == 6'b000011);

assign jr = (FuncCode == 6'b001000); //R-Type

assign jalr = (FuncCode == 6'b001001); //R-Type

......

assign RegWriteEN = (R & (addu | subu | jalr) ) | (lui | jal | addu | subu | ori | lw);

......

优点：即使指令数很多，代码也很简明，拼写错误易于发现。

缺点：不方便整理某条特定指令需要的各种信号的值。

1. 利用parameter(或宏定义)

parameter R = 6'b000000；

......

assign RegWriteEN = (OPCode == R & (FuncCode == addu | FuncCode == subu | FuncCode == jalr) ) | (OPCode == lui | OPCode == jal | OPCode == addu | OPCode == subu | OPCode == ori | OPCode == lw);

优点：可以将各种指令的OPCode进行批量导入，便于编程操作；

缺点：处理控制信号时，较复杂。

1. L0.T4
2. 未溢出时，ADD指令操作中的temp的第32位总是和第31位相等，进行的操作和ADDU完全一致，故二者在这种情况下等价；ADDI和ADDIU同理。
3. 优点：设计简单，操作方便，结构清晰；缺点：一个时钟周期只能进行一条指令，性能完全取决于处理器的时钟频率。
4. Jal是进栈前的最后一个操作，jal前要将当前运算中需要保存的量存入栈中，jr是出栈前的最后一个操作，jr前要将上一次进栈时的各个参数进行恢复。