**Project2 Logisim 完成单周期处理器开发**

**一：模块定义**

1. IFU模块定义：
2. 基本描述：

IFU 主要功能是完成取指令功能。IFU 内部包括 PC、IM(指令存储器)以及其他相关逻辑。IFU 除了能执行顺序取值令外，还能根据 BEQ 指令的执行情况决定顺序取值令还是转移取值令。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| IfBeq | I | 当前指令是否为 beq 指令标志。 1：当前指令为 beq 0：当前指令非 beq |
| Zero | I | ALU 计算结果为 0 标志。 1：计算结果为 0 0：计算结果非 0 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号。 1：复位 0：无效 |
| Instr[31:0] | O | 32 位 MIPS 指令 |

1. 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 | 取指令 | 根据 PC 从 IM 中取出指令。 |
| 3 | 计算下一条指令 地址 | 如果当前指令不是 beq 指令，则 PCPC+1 如果当前指令是 beq 指令，并且 zero 为 0，则 PCPC+1 如果当前指令是 beq 指令，并且 zero 为 1 ，则 PCPC+sign\_ext(当前指令 15..0) [注]PC 取地址为 4 字节，固低 2 位地址可以去除。 |

1. RegFile模块定义
2. 基本描述

RegFile的主要功能是完成读寄存器的值，并且将值输出；以及将所给的值写入寄存器。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RS1[4:0] | I | 被读取寄存器的寄存器号 |
| RS2[4:0] | I | 被读取寄存器的寄存器号 |
| RD[4:0] | I | 被写入寄存器的寄存器号 |
| WData[31:0] | I | 被写入的数据。 |
| RegWr | I | 是否可以写入数据。1：可以写入，0：不可以写入。 |
| Reset | I | 复位信号。 1：复位 0：无效 |
| Clk | I | 时钟信号。 |
| RD1 | O | RS1[4:0]寄存器中的数据 |
| RD2 | O | RS2[4:0]寄存器中的数据 |

1. 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 | 取寄存器中的数据 | 从寄存器堆中取指定寄存器中的数据。 |
| 3 | 改写寄存器中的数据 | 当RegWr为1时，将数据写入指定的寄存器中。 |

1. ALU模块定义
2. 基本描述

ALU的主要功能是用来计算算数运算结果。它根据不同的指令执行不同的功能。可以执行加法、减法、或运算、取立即数的高位四种不同的运算。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ALUctr[2：0] | I | ALU的控制端。00：ADD;01:SUB;10:OR;11:取立即数高位。 |
| A[31:0] | I | ALU输入的第一位操作数。 |
| B[31:0] | I | ALU输入的第二位操作数。 |
| C[31:0] | O | ALU的计算结果。 |
| Zero | O | ALU 计算结果为 0 标志。 1：计算结果为 0 0：计算结果非 0。 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | ADD | 对两个操作数执行加运算。 |
| 2 | SUB | 对两个操作数执行减运算。 |
| 3 | OR | 对两个操作数执行或运算。 |
| 4 | 取立即数高位 | 对第二个操作数取低16位作为高16位组成32位数。 |

1. DM模块定义
2. 基本描述

DM的主要功能是取数据存储器中存储的数据，还可以将自己的数据写入数据存储器中。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Address[4:0] | I | 传入DM的地址，用来写入或读取数据存储器的数据。 |
| Data[31:0] | I | 传入的数据，用来写入数据存储器。 |
| Clk | I | 时钟信号。 |
| MemWrite | I | 是否可以写入数据。1：可以写入，0：不可以写入。 |
| DataO[31:0] | O | 读取的数据存储器的数据。 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取数据 | 读取数据存储器中的指定地址中的数据。 |
| 2 | 写入数据 | 当MenWrite为1时，将传入的数据写入数据存储器。 |

1. control模块定义
2. 基本描述

control的主要功能是对每个指令产生所对应的控制信号的值。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| op[6:0] | I | 指令的31-26位 |
| funct[6:0] | I | 指令的0-5位 |
| RegDst | O | 写入寄存器的选择 |
| ALUSrc | O | ALU第二位输入信号的选择。0：选择寄存器堆中读取的第二个数据；1：选择符号扩展后的15位立即数。 |
| MemtoReg | O | 写入寄存器堆中数据的选择。0：选择ALU的计算结果；1：选择数据存储器中读取的数据。 |
| RegWrite | O | 是否将数据写入寄存器堆中。0：不写入；1：写入。 |
| MenWrite | O | 是否将数据写入数据存储器中。0：不写入；1：写入。 |
| nPC\_sel | O | 判断是否为分支指令。1：分支指令；0：不为分支指令。 |
| ExtOp | O | 判断扩展类型。 |
| ALUctr[2:0] | O | ALU的控制端。00：ADD;01:SUB;10:OR;11:取立即数高位。 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生各个控制信号 | 见模块接口中各控制信号的作用。 |
| 2 | 指令分类 | 将不同的指令分类。 |

1. EXT
2. 基本描述

按立即数按要求进行相应的扩展。符号扩展和0扩展。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[15：0] | I | 输入的16位立即数 |
| C | I | 扩展要求，0：0扩展；1：符号扩展。 |
| B[31：0] | O | 扩展后的32位数据 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 0扩展 | 对传入的立即数进行0扩展。 |
| 2 | 符号扩展 | 对传入的立即数进行符号扩展。 |

**二： 控制器设计**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 |  |  |  |  |  |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
|  | addu | subu | ori | lw | sw | beq | lui |
| RegDst | 1 | 1 | 0 | 0 | x | x | 0 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| MemtoReg | 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 | x | x | 0 | 1 | 1 | x | 0 |
| ALUctr[2:0] | 000 | 001 | 010 | 000 | 000 | 001 | 011 |

RegDst = add+sub;

ALUSrc = ori+lw+sw+lui;

MemtoReg = lw;

RegWrite = add+sub+ori+lw+lui;

MemWrite = sw;

nPC\_sel = beq;

ExtOp =lw+sw ;

ALUctr[0] = sub+beq+lui;

ALUctr[1] = ori+lui;

ALUctr[2]=0;

**三： 测试要求**

1. 测试程序的测试期望

寄存器$t3存储的是地址00000000，$s4中存储的地址是00000010。

内存地址00存00001111；01存00000010；10存00000001；12存00001010。

测试期望：寄存器$s2存储$t3中存储地址中的数字；寄存器$s2存储$t3中存储地址加4中的数字；$s1中的存储的为$s2,$s3的和；寄存器$s5存储$s4中存储地址中的数字；将寄存器$s6中的数字存储到$s4中存储地址加4的位置；寄存器$s7中存储寄存器$s2中数字和20异或的结果；寄存器$to存储$S4中存储地址加8中的数字；寄存器$t1存储$s4中存储地址加8中的数字；$t3中的存储的为$t0,$t1的和； 执行无符号加法运算，$t2中的存储的为$t0,$t1的和；寄存器$s7中存储寄存器$s2中数字和20异或的结果；将寄存器$s2中的数字存储到$t3中存储地址加8的位置；将寄存器$s3中的数字存储到$t3中存储地址加12的位置；将寄存器$t0中的数字存储到$s4中存储地址加12的位置；将寄存器$t1中的数字存储到$s4中存储地址加16的位置；取立即数20放到高16位给寄存器$t5。

1. 汇编指令以及注释

beq $0 , $0 , 1 beq指令执行成功，则指跳到pc+4+4；

lw $s2 , 0 ( $t3) 寄存器$s2存储$t3中存储地址中的数字

lw $s3 , 4 ($t3) 寄存器$s2存储$t3中存储地址加4中的数字

addu $s2 , $s2 , $0 执行无符号加法运算，$s2中存储的数字不变

addu $s3 , $s3 , $0 执行无符号加法运算，$s2中存储的数字不变

addu $s1 , $s2 , $s3 执行无符号加法运算，$s1中的存储的为$s2,$s3的和

lw $s5 , 0 ($s4) 寄存器$s5存储$s4中存储地址中的数字

sw $s6 , 4 ($s4) 将寄存器$s6中的数字存储到$s4中存储地址加4的位置

ori $s7 , $s2 , 20 寄存器$s7中存储寄存器$s2中数字和20异或的结果

lw $t0 , 8 ($s4) 寄存器$to存储$S4中存储地址加8中的数字

lw $t1 , 8 ($s4) 寄存器$t1存储$s4中存储地址加8中的数字

addu $t3 , $t0 , $t1 执行无符号加法运算，$t3中的存储的为$t0,$t1的和

subu $t2 , $t0 , $t1 执行无符号加法运算，$t2中的存储的为$t0,$t1的和

ori $s0 , $t3 , 20 寄存器$s7中存储寄存器$s2中数字和20异或的结果

sw $s2 , 8 ($t3) 将寄存器$s2中的数字存储到$t3中存储地址加8的位置

sw $s3 , 12($t3) 将寄存器$s3中的数字存储到$t3中存储地址加12的位置

sw $t0 , 12($s4) 将寄存器$t0中的数字存储到$s4中存储地址加12的位置

sw $t1 , 16($s4) 将寄存器$t1中的数字存储到$s4中存储地址加16的位置

lui $t5 , 20 取立即数20放到高16位给寄存器$t5

lui $t6 , 20 取立即数20放到高16位给寄存器$t6

**四：问答**

1. 请充分利用 Figure3 中的 X 可以将控制信号化简为最简单的表达式。

RegDst = add+sub;

ALUSrc = ori+lw+sw+lui;

MemtoReg = lw;

RegWrite =~（sw +beq）;

MemWrite = sw;

nPC\_sel = beq;

ExtOp = ~（ori+lui）;

ALUctr[0] = sub+beq+lui;

ALUctr[1] = ori+lui;

(设控制信号add为000，sub为001，or为010，取立即数放高位为011)

1. 对于 Figure5、Figure6 中的与或阵列来说，1 个 3 输入与门最终转化为 2 个 2输入与门，1 个 4 输入与门最终转化为 3 个 2 输入与门，依次类推。或阵列也类似计算。那么：
2. 请给出采用 Figure5、Figure6 中的方法设计的每个控制信号所对应的 2输入与门、2 输入或门、非门的数量。
3. 请与第 1 项对比，你更喜欢哪种设计方法。为什么？

RegDst:17个二输入与门，13个非门，1个二输入或门。

ALUSrc: 20个二输入与门，10个非门，3个二输入或门。

MemtoReg: 10个二输入与门，5个非门，1个二输入或门。

RegWrite: 32个二输入与门，21个非门，4个二输入或门。

MemWrite: 5个二输入与门，2个非门，1个二输入或门。

nPC\_sel: 5个二输入与门，5个非门，1个二输入或门。

ExtOp: 10个二输入与门，5个非门，1个二输入或门。

ALUctr[0]: 21个二输入与门，17个非门，2个二输入或门。

ALUctr[1]: 10个二输入与门，5个非门，1个二输入或门。

ALUctr[2]: 0个二输入与门，0个非门，0个二输入或门。

我更喜欢第一项设计方案，因为第一项设计方案使用的或门比较少，节省了资源。