计算机组成原理实验报告

1. 部件模块规格
2. IFU

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 处理器时钟信号 |
| Reset | I | 复位信号 |
| NextPC[4:0] | I | 下一个PC的值 |
| Instr[31:0] | O | 输出指令 |
| PC\_out[4:0] | O | 输出当前PC的值 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取指 | 从指令存储器取出将要执行的指令 |

1. GRF

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 处理器时钟信号 |
| RA1[4:0] | I | 读寄存器时第一个寄存器的编号（地址） |
| RA2[4:0] | I | 读寄存器时第二个寄存器的编号（地址） |
| WD[31:0] | I | 寄存器写入数据 |
| WA[4:0] | I | 写寄存器时的编号（地址） |
| RegWrite | I | 寄存器写使能信号 |
| RD1[31:0] | O | 读寄存器时第一个寄存器的输出数据 |
| RD2[31:0] | O | 读寄存器时第二个寄存器的输出数据 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | RD1输出RA1所寻址的寄存器中的数据  RD2输出RA2所寻址的寄存器中的数据 |
| 2 | 写寄存器 | 当时钟上升沿到来且RegWrite信号有效时，WD被写入WA所寻址的寄存器 |

1. ALU

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 参与ALU计算的第一个值 |
| B[31:0] | I | 参与ALU计算的第二个值 |
| ALUCtrl[1:0] | I | ALU功能的选择信号：  00：ALU进行加法运算  01：ALU进行减法运算  10：ALU进行或运算  11：ALU进行与运算 |
| Result[31:0] | O | ALU的计算结果 |
| Equal | O | 判断两数是否相等：  0：A!=B  1：A=B |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号加运算 | Result = A + B |
| 2 | 无符号减运算 | Result = A - B |
| 3 | 或运算 | Result = A | B |
| 4 | 与运算 | Result = A & B |
| 5 | 判断是否相等 | If(A == B) Equal=1 |

1. DM

（使用Logisim内置RAM）

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 控制器时钟信号 |
| Address[4:0] | I | 读或写时对应存储器时地址 |
| Load | I | 存储器读使能信号 |
| Input[31:0] | I | 写存储器时写入数据 |
| Store | I | 存储器写使能信号 |
| Data[31:0] | O | 读存储器时的输出 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读存储器 | Data输出存储器中地址Address存储的数据 |
| 2 | 写存储器 | 当时钟上升沿到来并且Store有效时，Input被写入存储器中地址Address的位置 |

1. EXT

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm[15:0] | I | 需要被扩展的16位立即数 |
| Extend[1:0] | I | 扩展方式选择信号：  00：符号扩展到32位  01：高位0扩展到32位  10：将立即数加载到高位，低位补0 |
| ext[31:0] | O | 扩展之后的输出数据 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 符号扩展 | 将imm进行符号扩展到32位 |
| 2 | 零扩展 | 将imm进行高位补0扩展到32位 |
| 3 | 加载到高位 | 将imm加载到高位，低位补0 |

1. 控制器模块规格

（1）控制器（Controller）

首先给出各个控制信号对应指令的真值表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | R(addu,subu) | sw | lw | beq | lui | ori |
| Opcode | I | 000000 | 101011 | 100011 | 000100 | 001111 | 001101 |
| RegDst | O | 1 | X | 0 | X | 0 | 0 |
| Branch | O | 0 | 0 | 0 | 1 | 0 | 0 |
| MemtoReg | O | 01 | X | 00 | X | 10 | 01 |
| MemWrite | O | 0 | 1 | 0 | 0 | 0 | 0 |
| MemRead | O | X | X | 1 | X | X | X |
| ALUOp | O | 11 | 00 | 00 | X | X | 10 |
| ALUSrc | O | 0 | 1 | 1 | 0 | X | 1 |
| RegWrite | O | 1 | 0 | 1 | 0 | 1 | 1 |
| Extend | O | X | 00 | 00 | 00 | 10 | 00 |

控制信号的意义，方便理解上述真值表

**RegDst**：控制寄存器堆的WD（写入地址端口）。0表示选择指令的25-21位为地址，1则表示选择20-16位为地址。

**Branch**：控制是否执行beq指令。Branch=0则是beq，此时若ALU的Equal端口为1，则执行PC=PC+1+imme操作；否则（Branch=0或者Equal=0）继续执行PC=PC+1

**MemWrite**：控制数据存储器的写使能。

**MemRead**：控制数据存储器的读使能。

**ALUOp**：控制ALU Controller的输出。ALUOp=00，控制器输出00；ALUOp=10,控制器输出10;ALUOp=11，控制器由FUNC决定。

**ALUSrc**：控制参与ALU运算的第二个操作数。ALU=0则为RD2，即寄存器堆第二个地址取出的数据；ALU=1则为扩展后的立即数。

**RegWrite**：控制寄存器堆的写使能

**Extend**：控制立即数扩展方式：00为符号扩展；01为高位补0扩展；10为加载到高位，然后低位补0的扩展。

（2）ALU控制器（ALU Controller）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 端口 | 方向 | 指令 | | | | | | |
|  |  | addu | subu | sw | lw | beq | lui | ori |
| ALUOp | I | 11 | 11 | 00 | 00 | X | X | 01 |
| Funct | I | 100001 | 100011 | X | X | X | X | X |
| operation | O | 00 | 01 | 00 | 00 | X | X | 10 |

**ALUOp**：输入，表示指令的类型（分为四类：R,sw/lw,beq/lui,ori）

**Funct**：仅对R指令有效，决定具体运算。

**Operation**：控制ALU运算。00表示加法，01表示减法，10表示或运算，11表示与运算。

（所以这里首先根据判断ALUOp判断指令类型，如果是非R类型，则operation直接等于ALUOp;如果是R类型，再根据Funct进行判断）

1. 测试程序与结果

我们先测试lui指令，这样等结束之后寄存器都不全是0，再进行测试比较方便

测试代码（遍历每一个寄存器）：

3c00a234

3c010001

3c020002

3c030003

3c040004

3c050006

3c06001a

3c07002d

3c081000

3c09a6b3

3c0a9999

3c0b0ff2

3c0ca222

3c0dbad3

3c0e4323

3c0f9876

3c10b678

3c112345

3c12a123

3c13a345

3c140001

3c159a2a

3c16bbbb

3c17cccc

3c18dddd

3c19ffff

3c1aee22

3c1b22ee

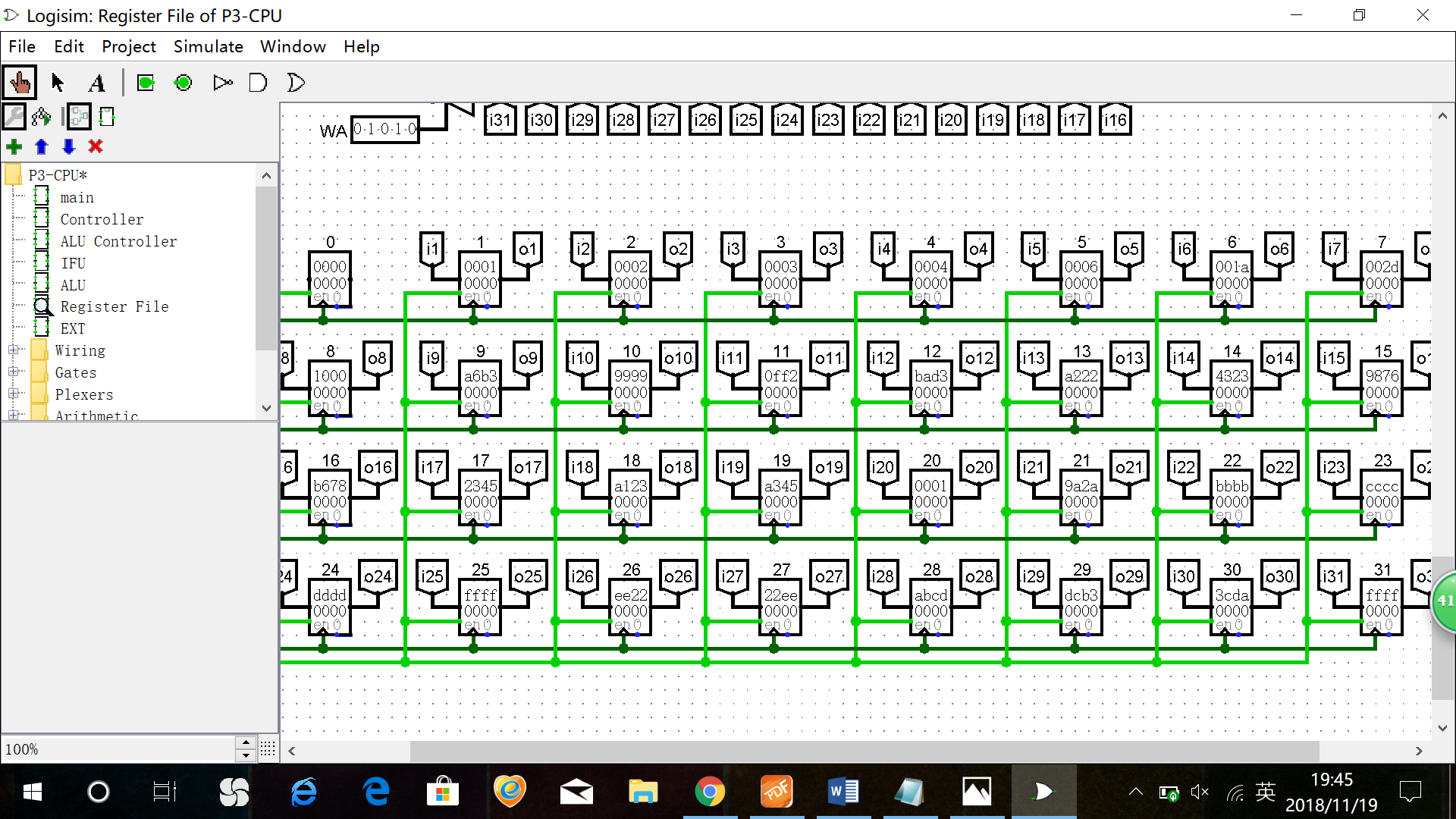
3c1cabcd

3c1ddcb3

3c1e3cda

3c1fffff

测试结果：



0号不改变，其他均符合要求。

下面测试sw指令（在上面寄存器的基础之上）

sw $3,0($0)

sw $4,8($0)

sw $8,12($0)

可以看到来自3，4，8号寄存器值0x00030000，0x00040000，0x10000000分别被存入存储器地址为0，2，3的位置。

再测试base!=0时候：

sw $20,4($1)

可以看到值0x00010000存入存储器地址为1位置。

在通过sw给DM存数后，便可以进行lw指令测试了：

lw $1,0($0)

lw $1,8($0)

lw $1,12($0)

$1相继存入0x00030000，0x00040000，0x10000000

再测试addu 与subu指令:

addu $1,$2,$3

addu $1,$2,$4

subu $1,$2,$3

subu $1,$3,$2

$1相继变化为0x00050000,0x00060000,0x00010000,0x00010000，说明无符号也没有问题。

最后是beq指令：只需观察PC 计数即可：

首先给1号，2号寄存器赋相同值ffff

3c01ffff

3c02ffff

然后开始测试：

10220001

10220005

1022000a

PC分别跳转2，6，11步

再测试不满足相等条件时：

10010005

10020005

发现并无跳转发生（PC按顺序执行）。

1. 思考题

T2:

1，首先由于存储器的地址对齐特点，PC储存值一定是4的倍数，换言之二进制下末两位为0，故换成30位可以减少硬件资源而并不影响功能。注意30位PC+1而不是+4.

2，合理，首先IM只需拥有读入功能，因为指令是一开始导入的，所有用ROM即可；GRF要求运算速度快，方便存取，故用寄存器实现；DM则要求读，写功能兼备，必须用RAM实现。

T3:

1,经分析知以下控制信号与func无关，故可以简化为：

RegDst=~op[0] & ~op[1] & ~op[2] & ~op[3] & ~op[4] & ~op[5]

ALUSrc=(op[0] & ~op[1] & op[2] & op[3] & ~op[4] & ~op[5])|( op[0] & op[1] & ~op[2] & ~op[4] & op[5])

MemtoReg=op[0] & op[1] & ~op[2] & ~op[3] & ~op[4] & op[5]

RegWrite=(op[0] & op[1] & ~op[2] & ~op[3] & ~op[4] & op[5])| (op[0] & ~op[1] & op[2] & op[3] & ~op[4] & ~op[5])| (~op[0] & ~op[1] & ~op[2] & ~op[3] & ~op[4] & ~op[5])

nPC\_Sel=~op[0] & ~op[1] & op[2] & ~op[3] & ~op[4] & ~op[5]

ExtOp=op[0] & op[1] & ~op[2] & ~op[4] & op[5]

2,

RegDst=~op[0] & ~op[1] & ~op[3] & ~op[4] & ~op[5]

ALUSrc=(op[0] & ~op[1] & op[2] & op[3] & ~op[4] & ~op[5])|( op[0] & op[1] & ~op[2] & ~op[4] & op[5])

MemtoReg=op[0] & op[1] & ~op[2] & & ~op[4] & op[5]

RegWrite=（op[0] & ~op[1] & op[2] & op[3] & ~op[4] & ~op[5]）|（~op[0] & ~op[1] & ~op[3] & ~op[4] & ~op[5]）|（op[0] & op[1] & ~op[2] & ~op[4] & op[5]）

nPC\_Sel=~op[0] & ~op[1] & op[2] & ~op[3] & ~op[4] & ~op[5]

ExtOp=op[0] & op[1] & ~op[2] & ~op[4] & op[5]

3,（以上面的控制信号命名为准）nop指令下，MemWrite和RegWrite全为0，即寄存器堆和存储器的值不会改变。而其他原件的各种变化毫无影响，不必关系。此外，nop指令本身没有任何功能，自然不需要考虑控制。因此从简练的角度，我们可以不加入nop。

T4：

1，假设DM有256MB容量，并且映射在0x30000000~0x3FFFFFFF区间。那么之需要把高4位地址与0x3进行比较，比较结果就是DM的片选信号。

2，

优点是由于形式验证技术是从理论上出发，借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，因此测试者不必考虑如何获得测试向量。同时这意味着形式验证是对指定描述的所以可能情况进行验证，更加严谨，全面。最后一点，形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早尽快地发现和改正电路设计中的错误，有可能缩短设计周期。

劣势之处在于，形式验证到目前为止仍然不能有效地验证电路的性能，如电路的时延和功耗等。