Project1 Logisim完成单周期处理器开发

# 设计说明

1. 处理器支持的指令集MIPS-Lite：addu，subu，ori，lw，sw，beq，lui。
   1. addu，subu可以不支持实现溢出。
2. 处理器为单周期设计。

# 设计要求

1. 顶层设计视图如Figure1所示。其部件包括：Controller(控制器)、IFU(取指令单元)、GPR(通用寄存器组，也称为寄存器文件、寄存器堆)、ALU(算术逻辑单元)、DM(数据存储器)、EXT(扩展单元)、多路选择器及splitter等。
   1. 顶层设计视图的顶层有效驱动信号包括且仅包括：clk、reset。
   2. 图中的其他字符均不是端口信号。

图示

描述已自动生成

Figure1 顶层设计

* 1. 采用模块化和层次化设计。整个设计文件目录结构如Figure2所示。

图形用户界面

低可信度描述已自动生成

Figure2 设计层次

1. IFU：内部包括PC、IM(指令存储器)及相关逻辑，如Figure3所示。

图示

描述已自动生成

Figure3 IFU

* 1. PC：用寄存器实现，宽度为30位，如Figure4所示。PC具有复位功能。

图示

描述已自动生成

Figure4 PC

* 1. IM：容量为32bit×32字，如Figure5所示，用ROM实现。

图示, 示意图

描述已自动生成

Figure5 IM

* 1. 说明：由于IM容量较小（32bit×32字），32（）字的地址仅需要5位用来寻址，因此采用2个对接的Splitter实现将PC的低5位地址与IM地址连接。

1. GPR：以32个32位具有写使能的寄存器为基础，辅以多路选择器，如图Figure6所示。

电脑屏幕的照片

描述已自动生成

Figure6 GPR

1. ALU：实现加法及减法时，不允许使用logisim内置的Adder及Subtractor。加法、减法实现必须以门电路为基础。

输入：数据X；数据Y；ALU控制信号ALUCtr[1:0]

输出：运算结果Result；零标志Zero

图示

描述已自动生成

Figure7 ALU

1. EXT：可以使用Logisim内置的Bit Extender。

输入：16-bit立即数、ExtOp[1:0]

输出：32-bit结果

功能：无符号/有符号扩展、16-bit扩展到32-bit的高16位

图示, 示意图

描述已自动生成

Figure8 EXT

1. DM：容量为32bit×32字，用RAM实现。
   1. DM应采用双端口模式，即设置RAM的“Data Interface”属性为“Separate load and store ports”。

图示, 示意图

描述已自动生成

Figure9 DM

1. 时钟源如下图所示。



Figure10 clk

* 1. 只有设置了时钟源，系统才能自动运行，从而让程序连续运行。

1. PC复位后初值为 0x0000\_3000，目的是与 MARS 的 Memory Configuration 相配合。

# 模块定义

1. 仿照下面给出的IFU模块定义，给出所有功能部件的模块定义。
   1. IFU、GPR、ALU、EXT、DM、Controller。

**（1）IFU模块定义**

1）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。

2）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 | O | 32位MIPS指令 |

3）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。 |

**（2）GPR模块定义**

1）基本描述

GPR包括了所有的寄存器，有两个读端口和一个写端口。寄存器的读输出总是对应于读寄存器号，不需要其他控制信号。但是写寄存器必须明确写使能控制信号。注意写操作是边沿触发的，所以所有的写操作的输入（要写的内容、寄存器号、写控制信号）必须在时钟边沿有效。因为寄存器的写入是边沿触发的，故可以在同一时钟周期内读出和写入同一寄存器：读操作读出以前写入的内容，而写入的内容在下一时钟周期才可读。寄存器号的输入都是5位的，数据线为32位。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1 | I | 读寄存器地址1 |
| A2 | I | 读寄存器地址2 |
| A3 | I | 写寄存器地址 |
| Data | I | 写入寄存器的数据 |
| clk | I | 时钟信号 |
| RegWrite | I | 寄存器写信号。 0：寄存器写使能无效 1：写使能有效 |
| Output1 | O | 寄存器读出的数据1 |
| Output2 | O | 寄存器读出的数据2 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | 根据读寄存器的地址A1/A2从寄存器中读数据，输出Ouoput1/Output2 |
| 2 | 写寄存器 | 如果寄存器的写使能有效，且在clk时钟信号触发边沿，则根据寄存器地址，将要写的数据写入指定地址的寄存器当中。读寄存器与写寄存器可以同时进行：读操作读出以前写入的内容，而写入的内容在下一时钟周期才可读。 |

**（3）ALU模块定义**

1）基本描述

ALU算数逻辑单元，是计算机的核心，在本次处理器设计中，ALU可以执行的算数运算包括加法、减法，可以执行的逻辑运算包括或操作。

本实验中，ALU由Adder，Subtractor，Orer以及一个多路选择器构成。多路选择器根据ALU控制信号ALUCtr，判断ALU应该进行哪种运算（加，减，或）。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| X | I | 参与运算的输入数据1 |
| Y | I | 参与运算的输入数据2 |
| ALUCtr | I | ALU控制信号  Switch (Ctr)  00: Result = A + B  01: Result = A - B  10: Result = A || B |
| Result | O | ALU运算结果 |
| ZERO | O | 运算结果是否为0的标志位，用于beq相等则分支指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | 若ALUCtr = 00，则执行加运算。把两个32-bit的输入数据相加，然后将结果Result输出。在本实验中，不支持溢出判断。 |
| 2 | 减运算 | 若ALUCtr = 01，则执行减运算。把两个32-bit的输入数据相减，然后将结果Result输出。 |
| 3 | 按位或 | 若ALUCtr = 10，则执行按位或运算。把两个32-bit的输入按位或，然后将结果Result输出。 |

**（4）EXT模块定义**

1）基本描述

EXT是符号拓展单元，有一个16位的输入，符号拓展为32位后输出。在lw取数、sw存数、beq相等则分支等指令中的立即数字段包含一个无符号/有符号二进制补码表示的16位数。为了将这个立即数字段加到一个32位的寄存器，计算机必须将这个16位的数转换成数值上相等的32位的数。

对于有符号数来说，这种方法就是将原有的16位数简单赋值到32位新数的低16位，其最高有效位（符号位）则以复制的方式填满新数的高16位，实现符号拓展。

对无符号数来说，将原有的16位数简单赋值到32位新数的低16位，用0填满新数的高16位。

对于lui指令，需要加载至高16位，将原有的16位数简单赋值到32位新数的高16位，低16位用0填充。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm16 | I | 16位立即数 |
| EXTOp | I | 符号拓展控制信号  Switch (EXTOp)  00: Imm32 = unsign 16 to 32 extend  01: Imm32 = sign 16 to 32 extend  10: Imm32 = extend to high 16 bit |
| Imm32 | O | 符号拓展结果输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 有符号拓展 | 将原有的16位数简单赋值到32位新数的低16位，其最高有效位（符号位）则以复制的方式填满新数的高16位，实现符号拓展。 |
| 2 | 无符号拓展 | 将原有的16位数简单赋值到32位新数的低16位，用0填满新数的高16位。 |
| 3 | 拓展至高16位 | 将原有的16位数简单赋值到32位新数的高16位，低16位用0填充。 |

**（5）DM模块定义**

1）基本描述

数据存储器单元是一个状态单元，两个输入为地址和所写数据，一个输出为读出的数据。写入数据需要写使能。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Address | I | 参与运算的输入数据1 |
| Input | I | 参与运算的输入数据2 |
| MemWrite | I | 数据存储器写使能 |
| clk | I | 时钟信号 |
| Reset | I | 复位 |
| Data | O | 读出的数据 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 根据地址Address从相应位置读出数据，Data为数据输出 |
| 2 | 写数据 | 如果MemWrite写使能有效，根据地址Address将数据Data写入存储器的相应位置 |

**（6）Controller模块定义**

1）基本描述

Controller是控制单元。控制单元以指令为输入，能够产生每个状态单元的写信号，每个多选器的选择信号和ALU的控制信号。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| func | I | 指令的func字段 |
| opcode | I | 指令的opcode字段 |
| ExtOP | O | 控制Extender的拓展方式 |
| ALUCtr | O | 控制ALU执行的运算 |
| PCSrc | O | 与ALU零输出相与决定PC是否跳转 |
| MemWrite | O | 数据存储器写使能 |
| RegWrite | O | 寄存器写使能 |
| MemToReg | O | 写入寄存器的数据来源：ALU/DM |
| ALUSrc | O | 第二个ALU操作数的来源 |
| RegDst | O | 写寄存器的目标寄存器号来源 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生控制信号 | 控制单元以指令为输入，能够产生每个状态单元的写信号，每个多选器的选择信号和ALU的控制信号。 |

# 控制器设计

1. MIPS-Lite指令集的单周期控制器真值表如下图所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号真值表 | | | | | | | | |
|  | func | 100001 | 100011 | n/a | | | | |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| 指令 | addu | subu | ori | lw | sw | beq | lui |
| RegDst | | 1 | 1 | 0 | 0 | X | X | X |
| RegWrite | | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| ALUSrc | | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| PCSrc | | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| MemRead | | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| MemWrite | | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| MemtoReg | | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ALUCtr[1:0] | | 00 | 01 | 10 | 00 | 00 | 01 | 00 |
| ExtOp[1:0] | | X | X | 00 | 01 | 01 | X | 10 |

* 1. 结合真值表，请给出数据通路每个功能部件的每个控制信号的布尔表达式。
  2. 表达式中只能使用“与、或、非”3种基本逻辑运算。
  3. 每个控制信号的表达式应该是指令opcode域与funct域的函数。
  4. 对于多位的控制信号(如ALUCtr)，应诸位给出其逻辑表达式。

addu = !op[0] & !op[1] & !op[2] & !op[3] & !op[4] & !op[5] & f[0] & !f[1] & !f[2] & !f[3] & !f[4] & f[5]

subu = !op[0] & !op[1] & !op[2] & !op[3] & !op[4] & !op[5] & f[0] & f[1] & !f[2] & !f[3] & !f[4] & f[5]

ori = !op[0] & !op[1] & !op[2] & !op[3] & !op[4] & !op[5]

lw = op[0] & op[1] & !op[2] & !op[3] & !op[4] & op[5]

sw = op[0] & op[1] & !op[2] & op[3] & !op[4] & op[5]

beq = !op[0] & !op[1] & op[2] & !op[3] & !op[4] & !op[5]

lui = op[0] & op[1] & op[2] & op[3] & !op[4] & !op[5]

RegDst = addu || subu

RegWrite = addu ||subu || ori || lw || lui

ALUSrc = ori || lw || sw || lui

PCSrc = beq

MemRead = lw

MemWrite = sw

MemtoReg = lw

ALUCtr[0] = subu || beq

ALUCtr[1] = ori

ExtOp[0] = lw || sw

ExtOp[1] = lui

1. 请在logisim中完成控制器设计。
   1. 控制器整体结构需要仿照Figure 11实现。

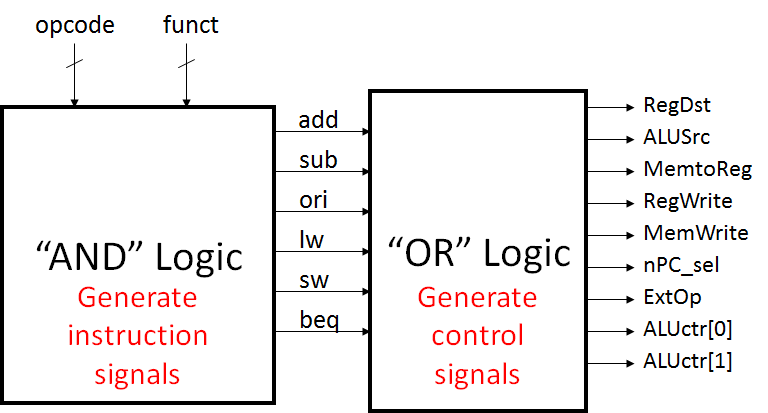


Figure 11 控制器基本结构

* 1. 控制信号必须仿照下图方式实现。

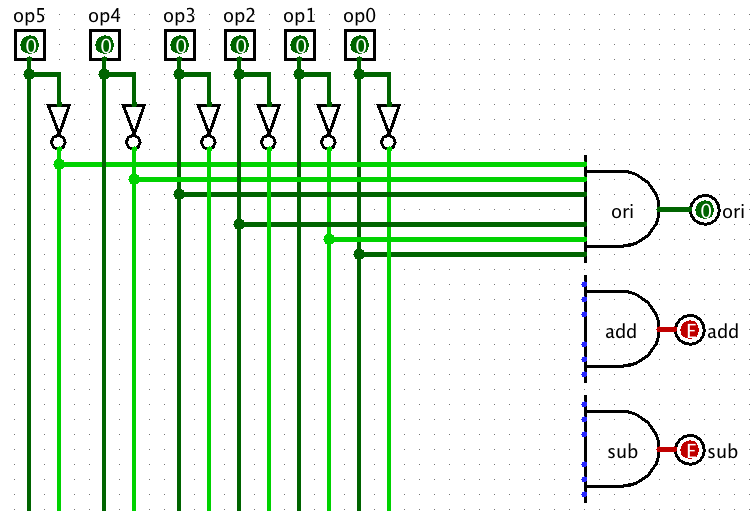


Figure12 与阵列：译码产生指令标识

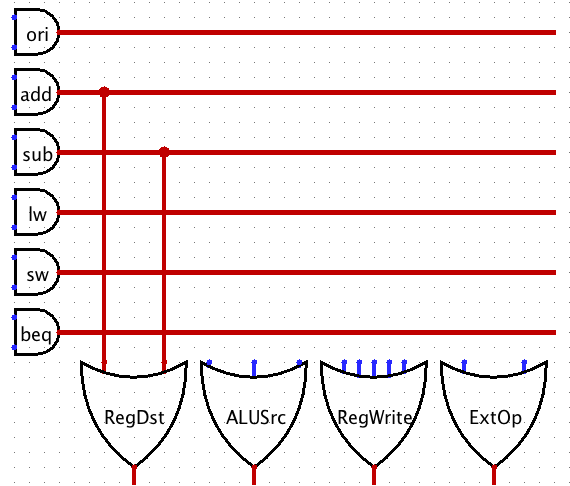


Figure13 或阵列：产生最终的控制信号

图示, 示意图

描述已自动生成

Figure14 控制器

# 测设要求

1. 所有指令都应被测试充分。
2. 构造1个至少20条以上指令的测试程序，并加载至IFU中运行通过。
   1. MIPS-Lite定义的每条指令至少出现1次以上。
   2. 演示时，测试程序必须已经通过IFU中的IM的“Load Image”加载完毕。

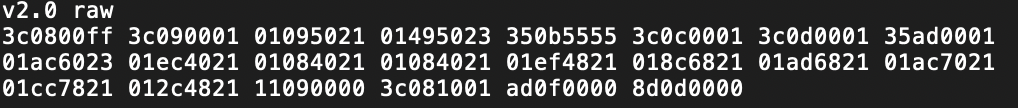


Figure15 二进制MIPS指令

1. 详细说明你的测试程序原理。
   1. 应明确说明测试程序的测试期望，即应该得到怎样的运行结果。
   2. 每条汇编指令都应该有注释。

|  |  |  |
| --- | --- | --- |
| 指令编号 | MIPS指令 | 含义 |
| 1 | lui $t0,255 | 立即数255加载至寄存器t0高16位 |
| 2 | lui $t1,1 | 立即数1加载至寄存器t1高16位 |
| 3 | addu $t2,$t0,$t1 | 寄存器t0和t1的值相加存在t2中 |
| 4 | subu $t2,$t2,$t1 | 寄存器t2和t1的值相减存在t2中 |
| 5 | ori $t3,$t0,21845 | 寄存器t0中的值与21845按位或存在t3中 |
| 6 | lui $t4,1 | 立即数1加载至寄存器t4高16位 |
| 7 | lui $t5,1 | 立即数1加载至寄存器t5高16位 |
| 8 | ori $t5,$t5,1 | 寄存器t5中的值与1按位或存在t5中 |
| 9 | subu $t4,$t5,$t4 | 寄存器t5和t4的值相减存在t2中 |
| 10 | addu $t0,$t7,$t4 | 寄存器t7和t4的值相加存在t0中 |
| 11 | addu $t0,$t0,$t0 | 寄存器t0和t0的值相加存在t0中 |
| 12 | addu $t0,$t0,$t0 | 寄存器t0和t0的值相加存在t0中 |
| 13 | addu $t1,$t7,$t7 | 寄存器t7和t7的值相加存在t1中 |
| 14 | addu $t5,$t4,$t4 | 寄存器t4和t4的值相加存在t5中 |
| 15 | addu $t5,$t5,$t5 | 寄存器t5和t5的值相加存在t5中 |
| 16 | addu $t6,$t5,$t4 | 寄存器t5和t4的值相加存在t6中 |
| 17 | addu $t7,$t6,$t4 | 寄存器t6和t4的值相加存在t7中 |
| 18 | addu $t1,$t1,$t4 | 寄存器t1和t4的值相加存在t1中 |
| 19 | beq $t0,$t1,lbbreak | 如果t0==t1,跳转到lbbreak标签 |
| 20 | lbbreak:lui $t0,4097 | 立即数4097加载至寄存器t0高16位 |
| 21 | sw $t7,0($t0) | 寄存器t7的内容存到t0为地址内存单元中 |
| 22 | lw $t5,0($t0) | 加载t0为地址的内存单元到寄存器t5中 |

MARS运行结果

1. 寄存器

表格

描述已自动生成

Figure16 MARS寄存器结果

1. 代码段

表格

中度可信度描述已自动生成

Figure17 MARS代码段结果

1. 数据段

表格

描述已自动生成

Figure18 MARS数据段结果

Logisim运行结果:

1. DM

图示, 示意图

描述已自动生成

Figure19 Logisim DM结果

1. RegFile

图示

描述已自动生成 图示

描述已自动生成

Figure20 Logisim RegFile结果

# 问答

1. 请充分利用控制信号真值表中的X可以将控制信号化简为最简单的表达式。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 控制信号真值表 | | | | | | | | |
|  | func | 100001 | 100011 | n/a | | | | |
| op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| 指令 | addu | subu | ori | lw | sw | beq | lui |
| RegDst | | 1 | 1 | 0 | 0 | X | X | X |
| RegWrite | | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| ALUSrc | | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| PCSrc | | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| MemRead | | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| MemWrite | | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| MemtoReg | | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ALUCtr[1:0] | | 00 | 01 | 10 | 00 | 00 | 01 | 00 |
| ExtOp[1:0] | | X | X | 00 | 01 | 01 | X | 10 |

RegDst = !op[0]

RegWrite = (!op[2] & op[1]) || (!op[5] & op[3]) || (op[5] & !op[3])

ALUSrc = op[0]

PCSrc = !op[3] & op[2]

MemWrite = op[3] & op[2]

MemToReg = op[5] & op[3]

ALUCtr[1] = op[3] & !op[1]

ALUCtr[0] = !op[3] & op[2]

ExtOp[1] = !op[5] & op[1]

ExtOp[0] = !op[2]

1. 对于Figure12、Figure13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
   1. 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。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2输入与门个数 | 2输入或门个数 | 非门个数个数 |
| RegDst | 11 | 1 | 12 |
| ALUSrc | 17 | 3 | 10 |
| RegWrite | 14 | 4 | 20 |
| MemWrite | 5 | 0 | 2 |
| ALUSrc | 17 | 3 | 10 |
| MemToReg | 5 | 0 | 3 |
| PCSrc | 5 | 0 | 5 |
| ALUCtr[1] | 5 | 03 | 3 |
| ALUCtr[0] | 11 | 1 | 11 |
| ExtOp[1] | 5 | 0 | 2 |
| ExtOp[0] | 11 | 1 | 5 |

* 1. 请与第17项对比，你更喜欢哪种设计方法。为什么

我更喜欢本次设计采用的方法。

第17项的方法是对每一个控制信号分配单独的与门、或门，因为它直接根据op[5:0]，func[5:0]的12位或6位逻辑信号表达式创建电路，没有指令的针对性，需要过多的原件。

本题中的方法首先根据op[5:0]，func[5:0]转换为相应的指令信号，再根据指令信号与控制信号的（真值表）关系，生成控制信号。这样可以用不同指令决定控制信号的输出，而不需要为某个控制信号单独设置指令的与门，这种方式使用的原件较少。

# 其他要求

1. 打包文件：Logisim工程文件、测试程序二进制文件、项目报告。
2. 截止时间：教师指定。
3. 本实验要求文档中凡是出现了【WORD】字样，就意味着该条目需要在实验报告中清晰表达。
4. 实验报告请按照《实验报告撰写规则》要求排版。

# 开发与调试技巧

1. 对于每条指令，请认真阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》！
   1. 如果测试时，你无法清楚的解释所要求的指令，测试成绩将减一档！
2. Figure1中Tunnel的用途是将具有相同name的tunnel连接在一起。Tunnel可以避免将图画的很乱。
3. Figure1中Probe的用途是显示被probed信号的值，便于调试。
4. Figure1中Splitter的用途是从某组信号中提取其中部分信号。例如，IFU输出32位指令，需要提取高6位(OpCode)和低6位(Funct)分别输入controller。
   1. splitter是有位序的！但字号太小，需要放大设计图(界面左下有比例设置)。
   2. 建议高位永远在上，低位永远在下
5. 如果你对于logisim内置的某个部件的端口不明白，请：
   1. 仔细阅读Help🡪Library Refrence关于该部件的描述。
   2. 放大logisim显示比例直至能清晰看到代表部件的各个端口的圆点，然后将鼠标停留相应的圆点上，就可以读取端口具体信息。
6. 建议先在MARS中编写测试程序并调试通过。
7. 片选信号就是对指令发出的数据存储器地址的高位分析。
   1. 假设DM有256MB容量，并且映射在0x3000\_0000～0x3FFF\_FFFF区间。那么只需要把高4位地址与0x3进行比较，比较结果就是DM的片选信号。
   2. 为了实现片选，你需要用基本逻辑门搭2个片选信号生成逻辑，一个输出至IM，一个输出至DM。
   3. Logisim内置的RAM有片选信号！
8. 提示：你可以考虑增加7段数码管等输入输出来让你的测试结果更加直观。
   1. 本条非必做要求。
   2. 7段数码管也需要类似片选等信号，其工作原理与第31项类似。