Project1 Logisim完成单周期处理器开发 实验报告

# 设计说明

1. 处理器支持的指令集MIPS-Lite：addu，subu，ori，lw，sw，beq，lui，j。
   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。

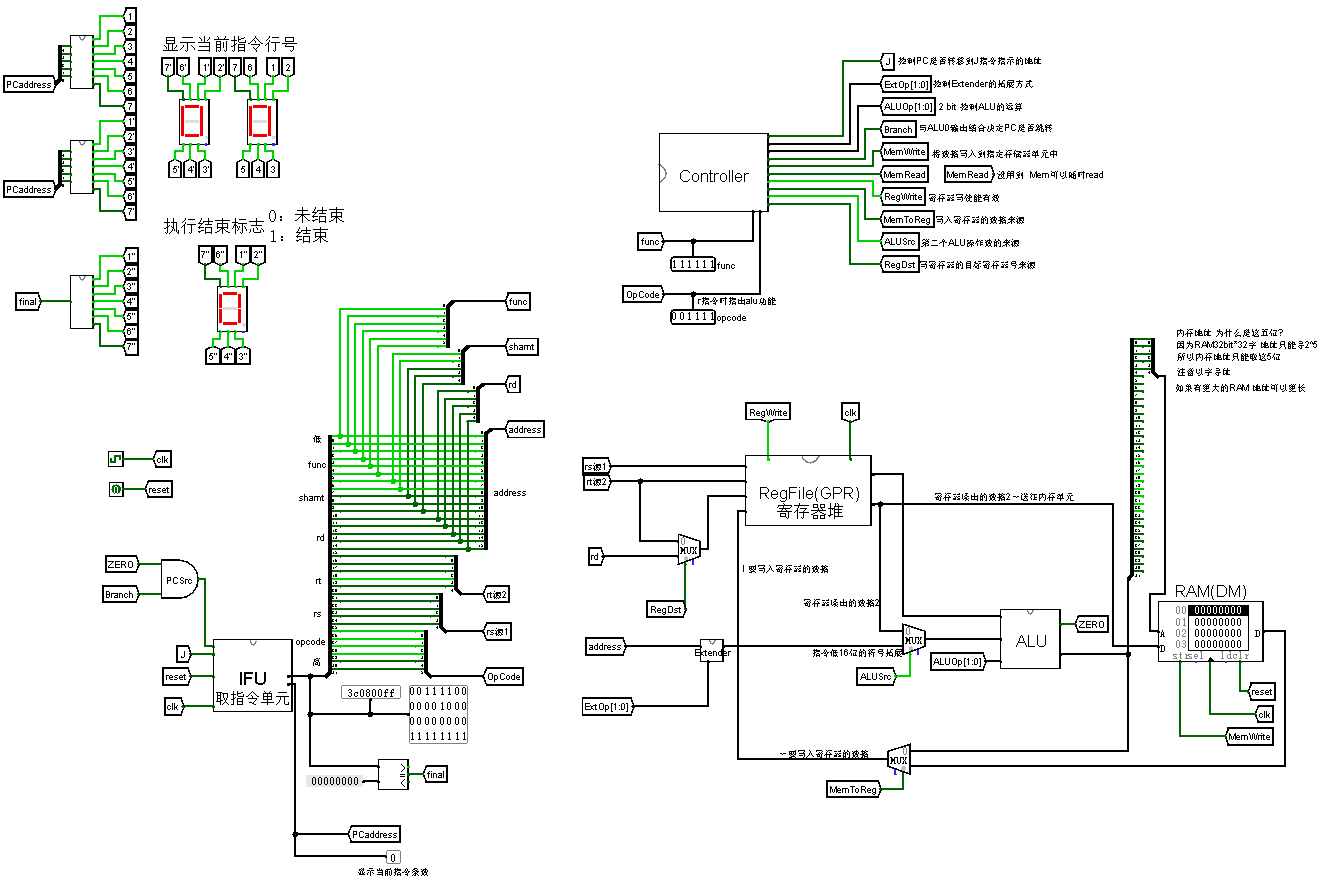


Figure1 顶层设计

* 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。

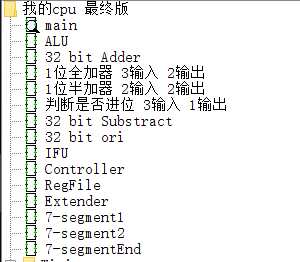


Figure2 设计层次

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

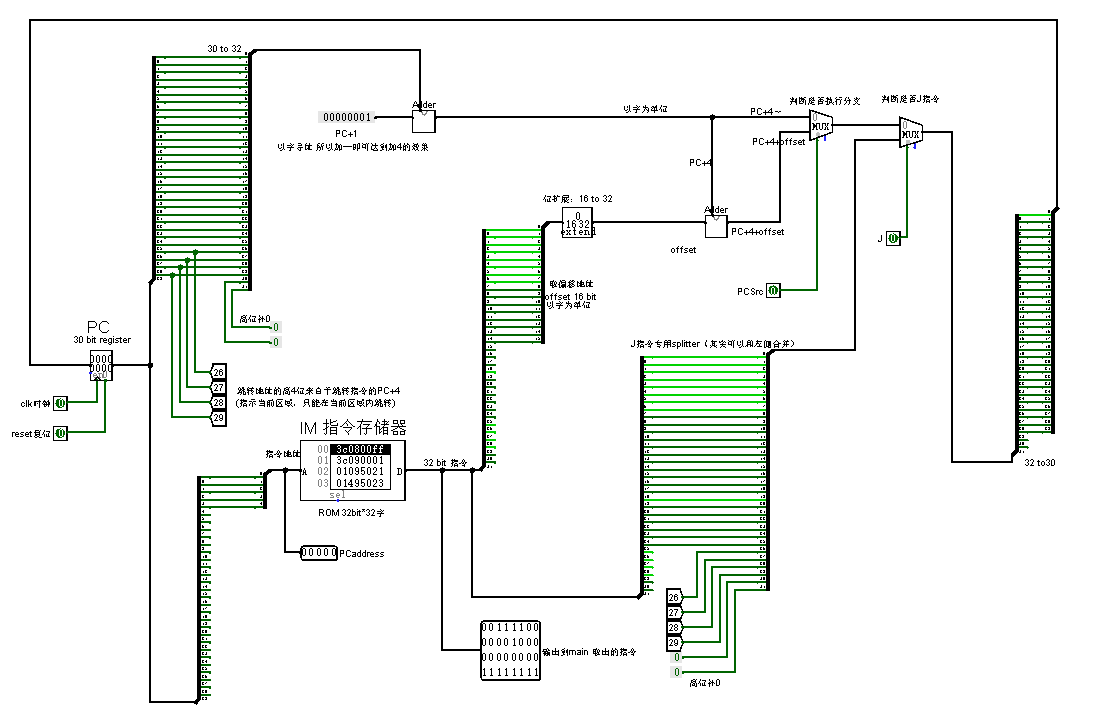


Figure3 IFU

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

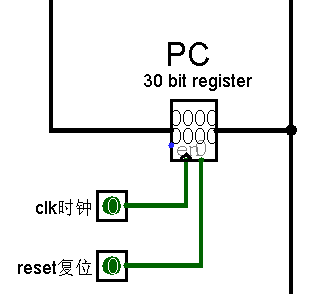


Figure4 PC

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

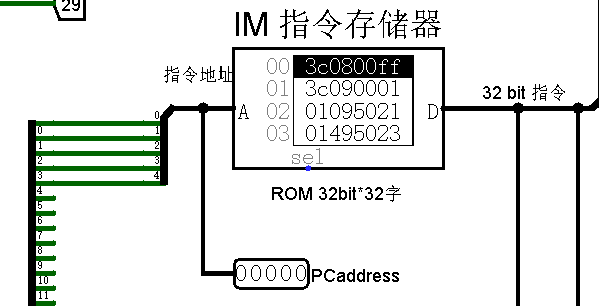
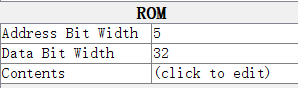
 

Figure5 IM

* 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，图Figure6。

**输入：**

读寄存器地址A1，A2；写寄存器地址A3；写入寄存器的数据；寄存器写使能；时钟clk

**输出：**

读出的数据D1,D2

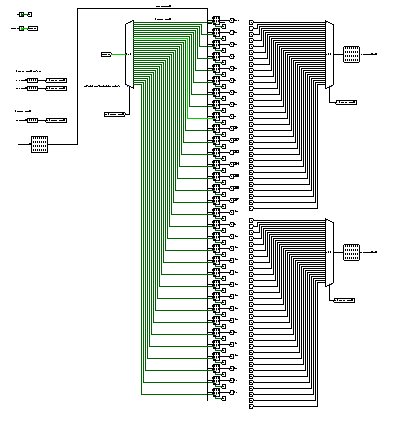


Figure6 GPR

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A；数据B；ALU控制信号ALUop

**输出：**

运算结果；零标志

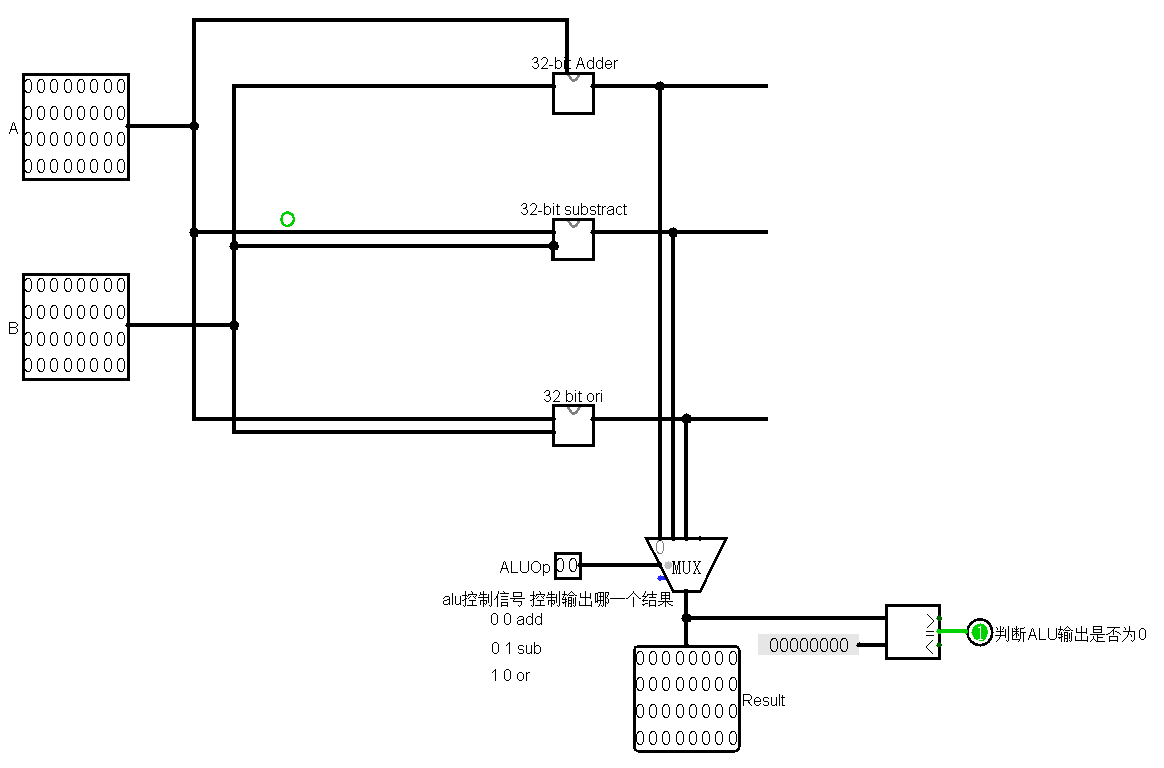


Figure7 ALU

1. EXT：可以使用logisim内置的Bit Extender。

**功能：**

16-bit to 32-bit有符号/无符号拓展；16-bit拓展到32-bit的高16位

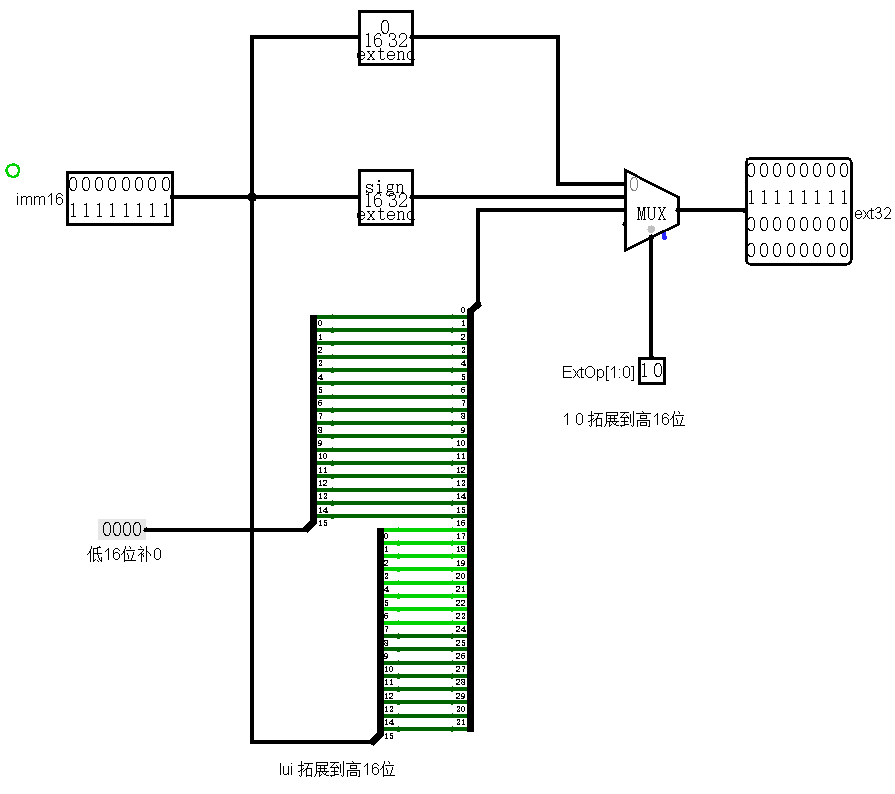


Figure8 EXT

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”。

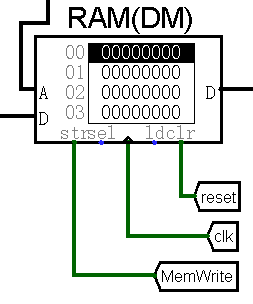
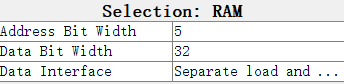
 

Figure9 DM

1. 时钟源如下图所示。

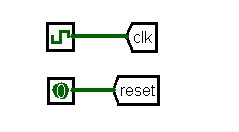


Figure10 clk

* 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。

# 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[31:0] | 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 |
| WD | 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 |
| D1 | O | 寄存器读出的数据1 |
| D2 | 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D1/D2 |
| 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32-bit adder，32-bit subtract，32-bit ori以及一个多路选择器构成。多路选择器根据ALU控制信号ALUop，判断ALU应该进行哪种运算（加，减，或）。

首先从构造1位半加器、1位全加器开始，然后构造一个判断是否进位的逻辑单元，将其组合在一起，构成32位的算数运算单元。在构建32位的算数运算单元时应该注意：第0位不需要考虑进位，只有两个输入，而其余31位有3个输入都需要考虑低位是否有进位。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 参与运算的输入数据1 |
| B | I | 参与运算的输入数据2 |
| ALUop | I | ALU控制信号  Switch (ALUop)  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ALUop = 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ALUop = 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ALUop = 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 |
| Addr | I | 参与运算的输入数据1 |
| DataIn | 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 |
| DataOut | 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Addr从相应位置读出数据，DataOut为数据输出 |
| 2 | 写数据 | 如果Memwrite写使能有效，根据地址Addr将数据DataIn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 |
| J | O | 控制PC是否转移到J指令的地址 |
| ExtOP | O | 控制Extender的拓展方式 |
| ALUOp | O | 控制ALU执行的运算 |
| Branch | O | 与ALU零输出结合决定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。



Figure11 控制信号真值表

* 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。

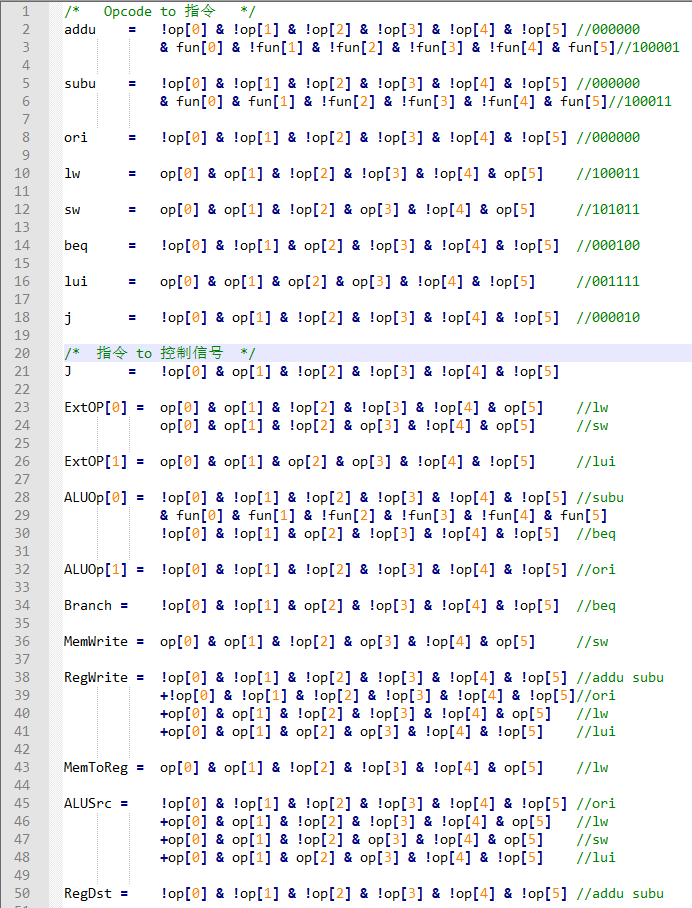


Figure11 逻辑表达式

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 12实现。

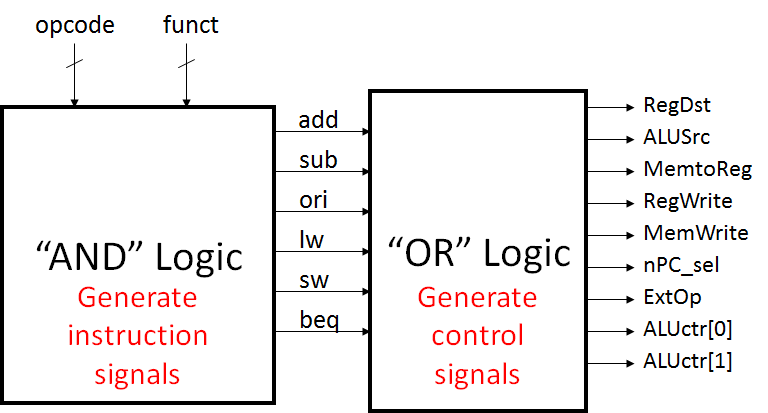


Figure 12 控制器基本结构

* 1. 控制信号必须仿照下图方式实现。

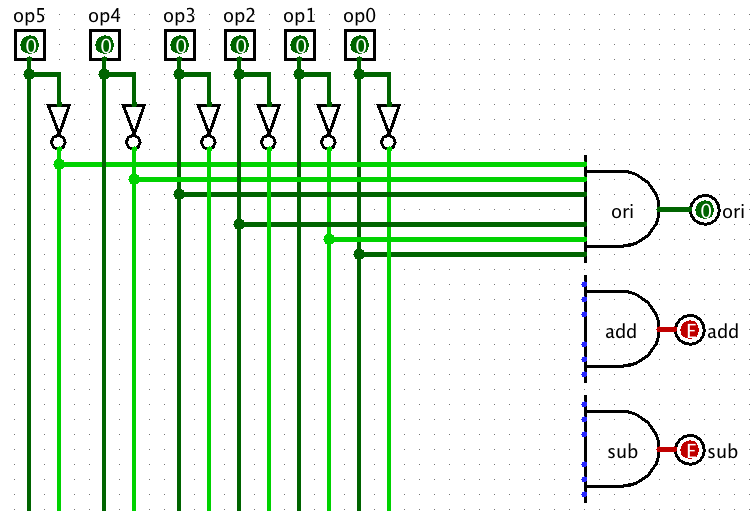


Figure13 与阵列：译码产生指令标识

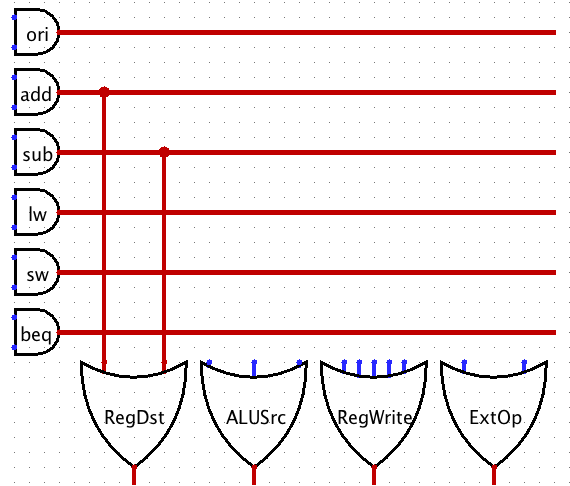


Figure14 或阵列：产生最终的控制信号

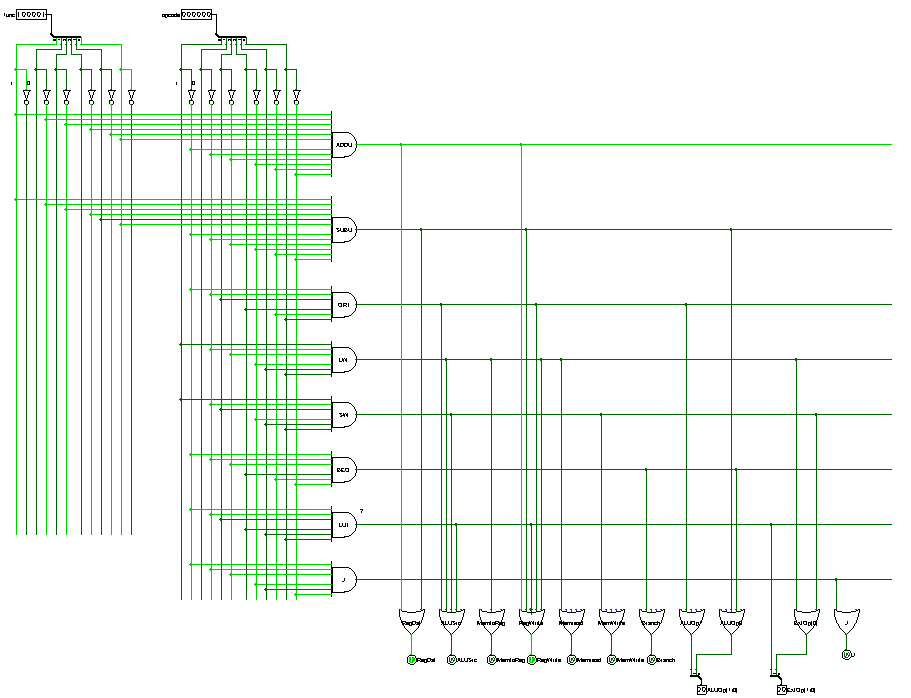


Figure15 控制器

# 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。

**测试指令：**

3c0800ff 3c090001 01095021 01495023 350b5555 3c0c0001 3c0d0001 35ad0001 01ac6023 01ec4021 01084021 01084021 01ef4821 018c6821 01ad6821 01ac6821 01ac7021 01cc7821 012c4821 11090001 0810000f 3c080000 ad0c0000 8d0d0000

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。

**指令说明：**



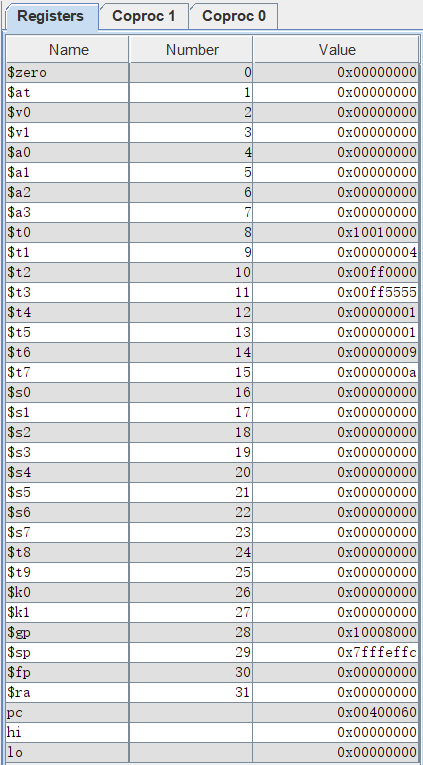
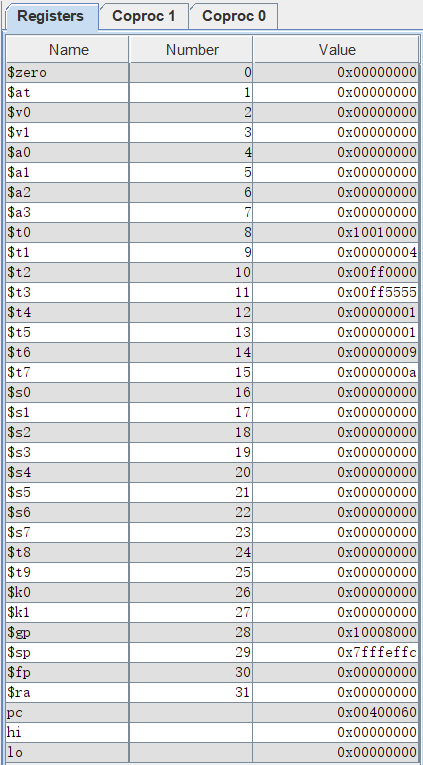
\*注：MARS中数据段从地址10010000开始，Logisim数据寄存器地址从0开始，因此在使用logisim模拟时，需要注意地址需要手动更改（如图灰色所示）。

**指令、机器码、十六进制码对照**

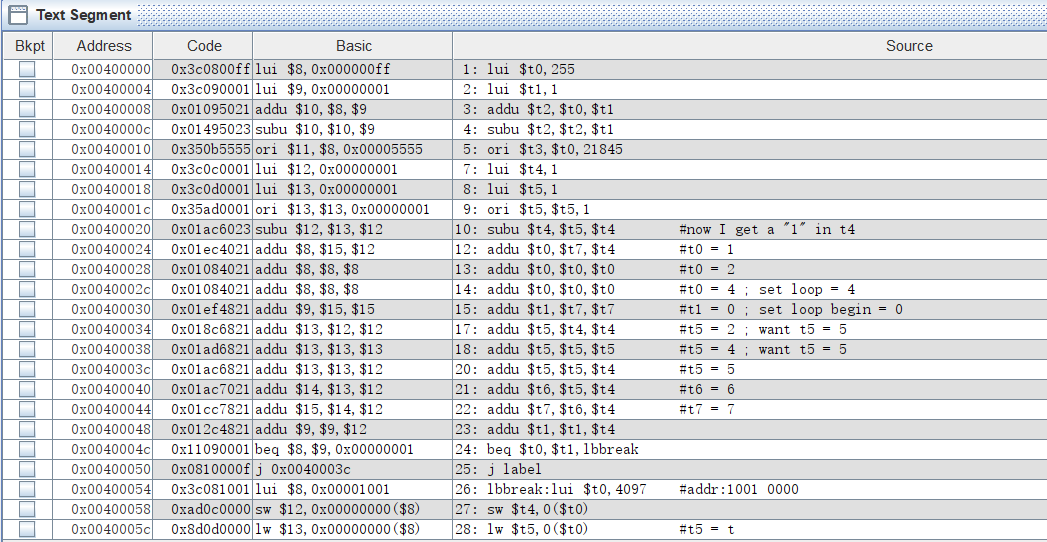


**MARS运行结果**

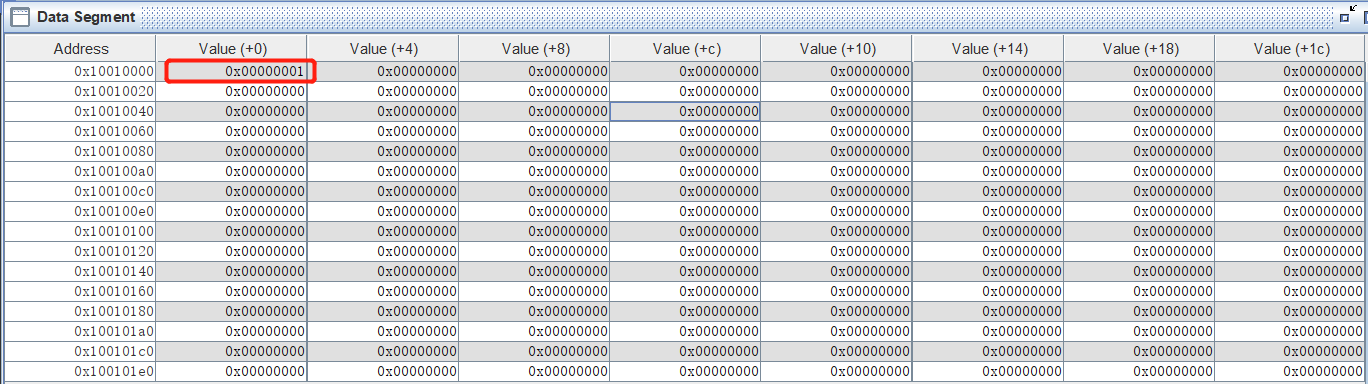
（1）寄存器

（2）代码段

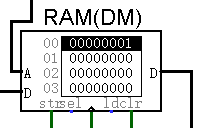


（3）数据段

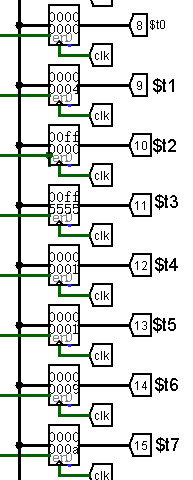


**Logisim运行结果**

RAM：



Register：



# 问答

1. 请充分利用Figure11中的X可以将控制信号化简为最简单的表达式。



最简表达式：

J = op1 ~op0

ExtOP[0] = ~op2

ExtOP[1] = ~op5 op1

ALUOp[0] = ~op3 op2

ALUOp[1] =op3 ~op1

Branch = ~op3 op2

MemWrite = op3 ~op2

RegWrite = ~op2 ~op1 + ~op5 op3 + op5 ~op3

MemToReg = op5 ~op3

ALUSrc = op0

RegDst = ~op0

1. 对于Figure13、Figure14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. 请与第17项对比，你更喜欢哪种设计方法。为什么



我认为本题中的方法更好。

方法一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. 时间要求：实验指导教师指定。
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。

# 实验测试要求

1. 实验成绩由下列部分组成：回答问题、MIPS-Lite处理器正确性、增加新指令后的处理器正确性等。
2. 实验测试时，你需要展示你的设计并证明其正确性。
3. 实验指导教师会临时增加1～2条指令，你需要在规定的时间内完成对原有设计的修改，并通过实验指导教师提供的测试程序。

# 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。
   1. 注意MARS中的“Settings🡪Memory Configuration”只能配置指令存储器起始地址为0地址，而不能将指令存储器和数据存储器的起始地址均配置为0地址！
   2. 由于logisim设计中的DM起始地址为0，因此请仔细观察所用到的指令，在把MARS中调试通过的二进制码导出后，你可能需要手工修改指令码中的数据偏移。
   3. 提示：事实上，在现代主流计算机中，数据存储器和指令存储器的起始地址不应该重叠。但在本设计中，由于采用分离存储器设计方案，因此可以暂时忽略这一点。
7. 当然，如果你能再自学一点点存储器译码的知识，那么只需再增加一个DM片选信号，一切都搞定了(就不需要再考虑第30.b)了)。
   1. 片选信号就是对指令发出的数据存储器地址的高位分析。
   2. 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。
   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。