

**计算机组成原理-实验报告**

院 系 软件学院

专业班级 软件2202班

姓 名 郑德凯

学 号 U202217216

指导教师 黄浩/胡雯蔷

2024年 12月 1 日

**目 录**

[1 课程实验概述 1](#_Toc17235)

[1.1 实验目的 1](#_Toc26153)

[1.1.1 运算器实验 1](#_Toc32230)

[1.1.2 存储系统实验 1](#_Toc6434)

[1.2 实验内容 1](#_Toc13957)

[1.3 实验环境 2](#_Toc3526)

[2 运算器组成实验 2](#_Toc17334)

[2.1 八位串行可控加减法器电路设计 2](#_Toc20733)

[2.2四位先行进位电路CLA74182 4](#_Toc13650)

[2.3四位快速加法器 7](#_Toc22408)

[2.4十六位快速加法器 8](#_Toc26681)

[2.5 32位快速加法器 10](#_Toc24925)

[2.6 32位MIPS运算器 11](#_Toc183)

[3 存储系统综合实验 15](#_Toc7428)

[3.1 存储扩展实验 15](#_Toc18683)

[3.2 MIPS寄存器文件设计 17](#_Toc12972)

[4 心得体会 21](#_Toc7242)

# 1 课程实验概述

1.1 实验目的

1.1.1 运算器实验

熟悉 Logisim 软件平台。

掌握运算器基本工作原理

掌握运算溢出检测的原理和实现方法；

理解有符号数和无符号数运算的区别；

理解基于补码的加/减运算实现原理；

熟悉运算器的数据传输通路。

1.1.2 存储系统实验

熟悉 Logisim 软件平台；

熟悉 ROM、RAM 存储器的使用；

掌握存储器字扩展，位扩展的基本原理；为 MIPS CPU 设计功能部件---

寄存文件；

1.2 实验内容

2.1 八位串行可控加减法器电路设计

2.2 四位先行进位电路

2.3 4 位快速加法器设计

2.4 16 位快速加法器设计



2.5 32 位快速加法器设计

2.6 32 位 MIPS 运算器设计

3.1 存储扩展实验

3.2 MIPS 寄存器文件设计4

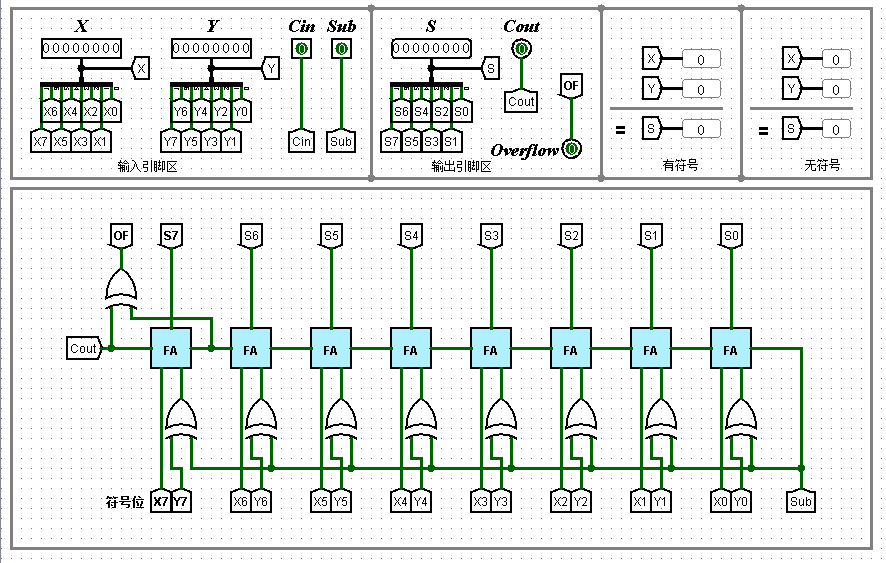
1.3 实验环境

Logisim是一款旨在帮助学习者设计和模拟数字逻辑电路的教育软件，具有直观且易于操作的特点。作为一款基于Java开发的应用程序，Logisim能够在任何支持Java环境的平台上运行，因此特别适合学生用来学习数字电路的设计与实现。Logisim的核心功能之一是提供图形化界面来设计和展示中央处理单元（CPU）的架构。此外，软件还支持多种电路分析模型，如转换电路、布尔表达式、真值表等，为用户提供了丰富的工具来辅助电路设计与分析。Logisim还允许用户将小规模的电路模块化，作为大型电路的组成部分，极大地提高了电路设计的可重用性与灵活性。

# 2 运算器组成实验

2.1 八位串行可控加减法器电路设计

**1) 电路图**



**2) 设计分析与说明：**

* 将8个一位全加器FA的进位链串联即可得到8位加法器，由于补码符号位也可以参与运算，所以此电路既可以用于有符号数运算，也可以用于无符号数运算，但二者在溢出检测上有一定区别，这里OF的判定以有符号数加法运算是否溢出为标准。
* 溢出检测：根据运算过程中，最高数据位的进位与符号位的进位位是否一致进行检测。V= Cd xor Cf。
* sub = 0，执行减法操作。0和二进制数异或运算得到其本身，然后通过一位全加器FA执行加法运算。
* sub = 1，执行减法操作。1和二进制数异或运算相当于对二进制数进行取反操作，然后将sub = 1，传入FA进行+1操作。（由[y]补求[-y]补，全部位取反后加一）。
* 高位进位的产生依赖于低位进位的输入，串行进位加法器的速度较慢。
* 输入：操作数1X的8位数据X7-X0，操作数2Y的八位数据Y7-Y0。最低位进位Cin，加减法控制项Sub。
* 输出：运算结果S的八位数据S7-S0，最高位进位Cout，有符号运算溢出判断OF。

3) 实验结果的记录与分析：

1. x=0000 0000，y=0000 0001，sub=0；s=0000 0001，Cout=0，OF=0
2. x=0000 1000，y=0000 0101，sub=1；s=0000 0011，Cout=1，OF=0
3. x=0100 1000，y=0100 0001，sub=0；s=1000 1001，Cout=0，OF=1
4. x=0000 0111，y=1000 0111，sub=1；s=1000 0000，Cout=0，OF=1

分析第1个输出结果 ：

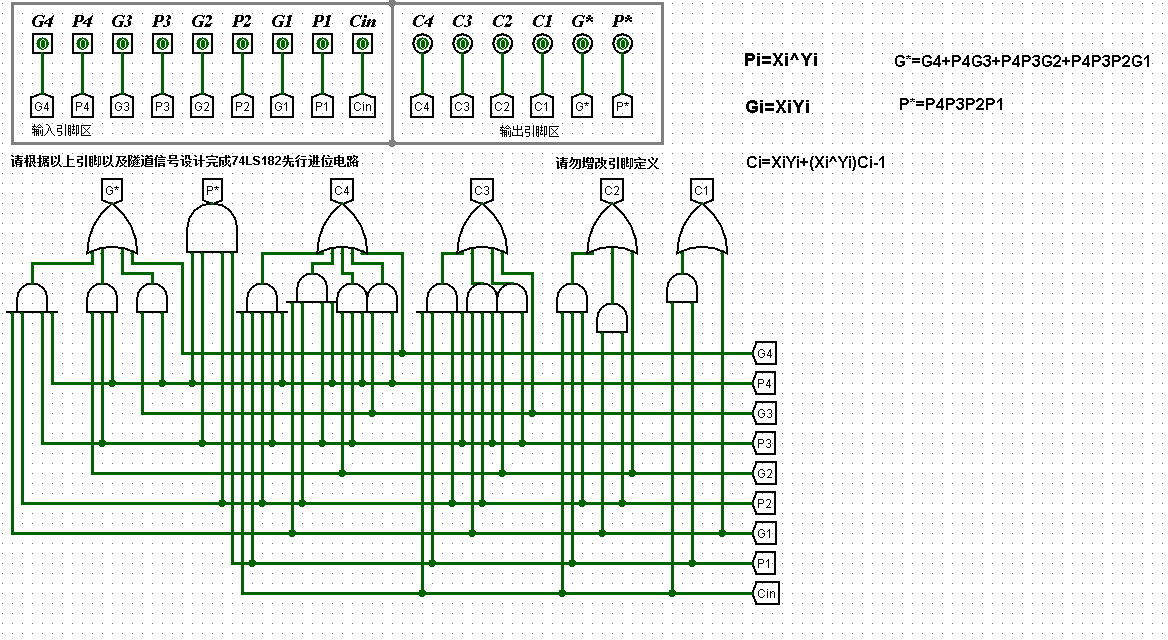
sub = 0 ，两个正数执行x + y操作。输出结果为s=0000 0001，没有发生溢出，最高位的进位为0。

4) 操作步骤及顺序：

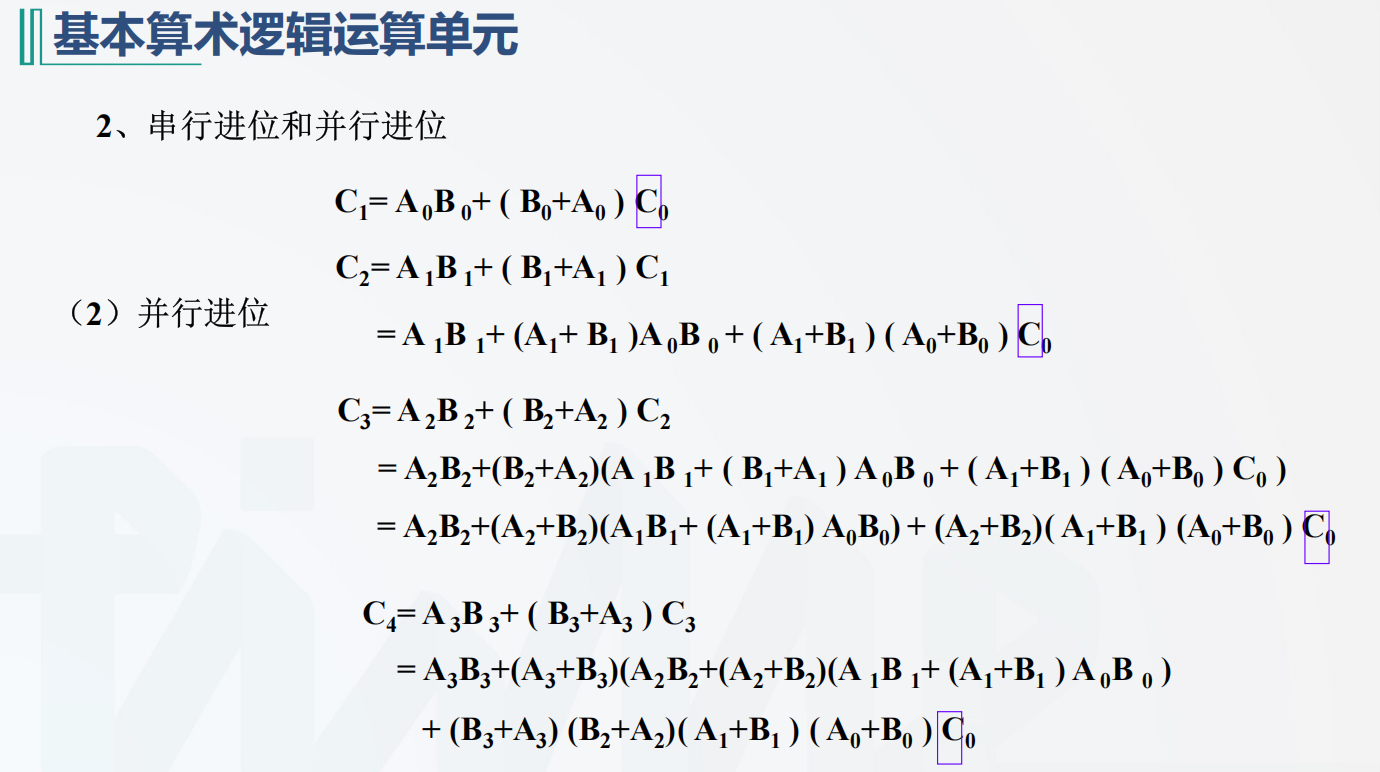
* 根据sub确定是执行减法操作还是加法操作。
* 将操作数的8位数据的每一分别送入一位全加器FA中执行加法操作。
* 得出结果

2.2四位先行进位电路CLA74182

**1) 电路图**



**2) 设计分析与说明：**



1. A3-A0,B3-B0分别是操作数1A和操作数2B的四位数据，C1、C2、C3、C4分别是从低至高位的进位数据，C0是低位进位输入，可以以此代入展开。
2. 因此C1，C2，C3,C4的计算便不需要彼此依赖，而是可以独立地根据A3-A0，B3-B0以及C0计算出来。根据这样的思路，便可以实现“四位先行进位电路”。
3. Gi=Ai-1 Bi-1

Pi=Ai-1 + Bi-1

G=G4 + P4G3 + P4P3G2 + P4P3P2G1

P=P4P3P2P1

1. 如C2 = G2 +P2G1 + P2P1C0。先P2P1C0进行AND运算，P2G1进行AND运算，再将它们的结果与G2进行OR运算。

输入：进位产生函数G4、G3、G2、G1，进位传递函数P4、P3、P2、P1以及低位进位Cin

输出：四个进位C4、C3、C2、C1,进位产生函数G，进位传递函数P

3) 实验结果的记录与分析：

输入序列G4 P4 G3 P3 G2 P2 G1 P1 Cin，输出序列C4 C3 C2 C1 G\* P\*

1. 输入：0000 0001 1 输出：0001 00
2. 输入：0000 0111 0 输出：0011 00
3. 输入：1111 0000 0 输出：1100 10
4. 输入：1111 1110 0 输出：1111 10

分析输出结果 ：

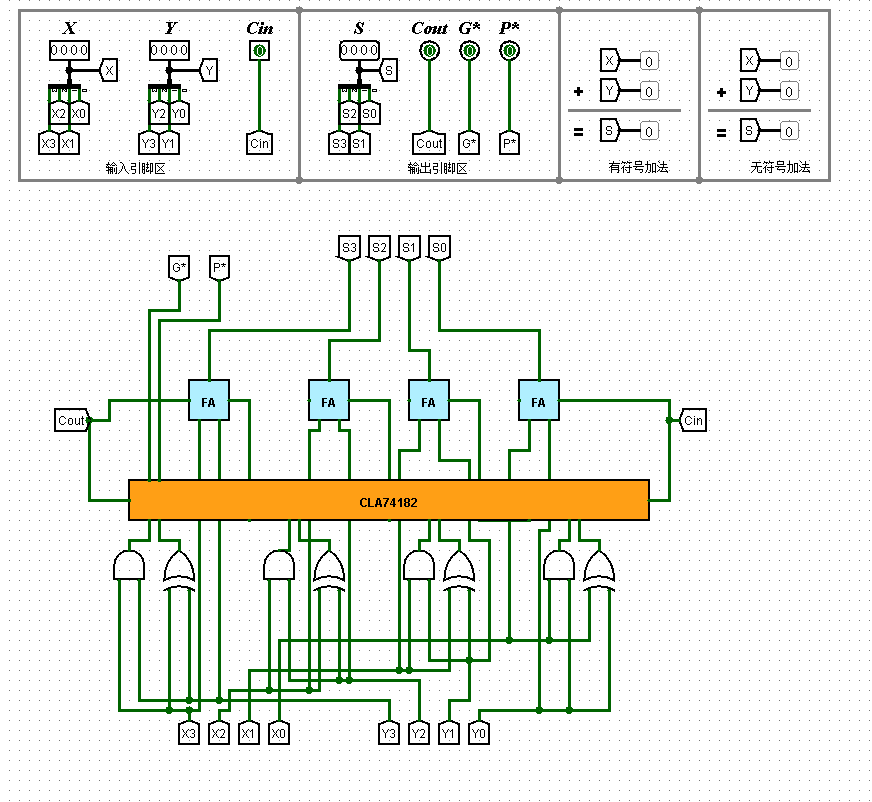
C1，C2，C3, C4的计算不需要彼此依赖，而是可以独立地根据A3-A0，B3-B0以及C0计算出来。

4) 操作步骤及顺序：

1. 输入序列G4 P4 G3 P3 G2 P2 G1 P1 Cin
2. 根据公式分别算出C4 C3 C2 C1 G\* P\*。以C2 = G2 +P2G1 + P2P1C0为例：先P2P1C0进行AND运算，P2G1进行AND运算，再将它们的结果与G2进行OR运算。
3. 输出结果C4 C3 C2 C1 G\* P\*

2.3四位快速加法器

1) 电路图



2) 设计分析与说明：

需要一个CLA74182作为来并行计算进位值，然后只需要4个一位全加器FA实现加法运算即可。

输入：X3X2X1X0, Y3Y2Y1Y0, Cin

输出：运算结果S的四位数据S3-S0，最高位进位Cout，p\*,G\*。

3) 实验结果的记录与分析：

输入：x = 1000 , y = 0100, Cin = 0 ; 输出：s = 1100, Cout = 0，p\* = 0, G\* = 0

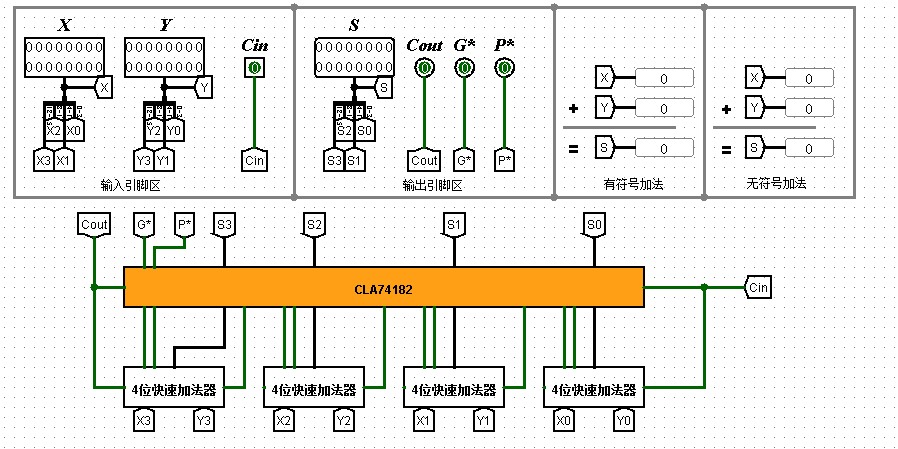
分析：先对Xn、Yn分别进行AND和OR运算，送入CLA74182，再由此算出S3-S0

4) 操作步骤及顺序：

* 先求Gi=Xi Yi, Pi=Xi + Yi
* 根据CLA74182得出的C4 C3 C2 C1作为进位位， 分别与对应的每一位X和Y使用FA进行加法操作
* 得出运算结果四位数据S3-S0，以及最高位进位Cout，p\*,G\*。

2.4十六位快速加法器

**1) 电路图**



**2) 设计分析与说明：**

需要一个CLA74182作为来并行计算进位值，然后只需要4个四位快速加法器实现4位数的运算即可。

**3) 实验结果的记录与分析：**

输入：X = 0000 1100 0000 1100, Y = 1000 0100 1000 0100, Cin = 0;

输出：S = 1001 0000 1001 0000, Cout = 0, p\* = 0, G\* = 0

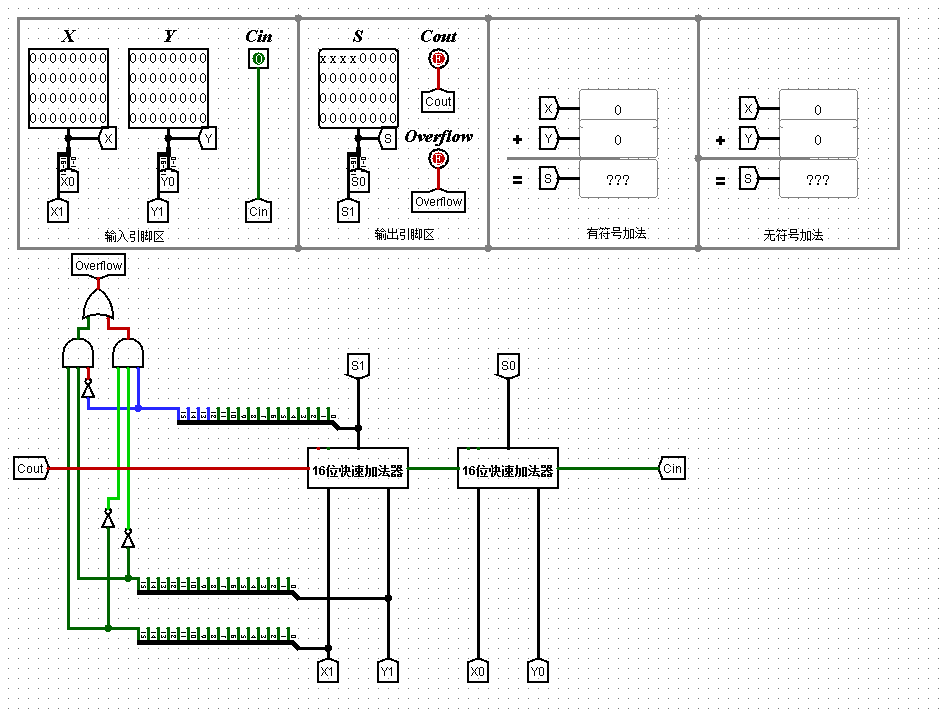
分析：将输入的32位数据分成4个4位数据分别送入，利用4位加法器分别算出S0-S3，利用CLA74182算出p\* 和G\*，结果正确，与预期相 符合。

**4) 操作步骤及顺序：**

* 将输入的32位数据分成4个4位数据分别送入4位快速加法器中
* 结合Cin的值得出S0-S3
* 将每一个4位快速加法器产生的p\* 和G\*送入CLA74182，算出最终的p\* 和G\*
* 得出最终结果

2.5 32位快速加法器

1)电路图



2) 设计分析与说明：

* 需要一个CLA74182作为来并行计算进位值，然后只需要2个十六位快速加法器实现16位数的运算即可。
* 溢出检测方法1：根据操作数和运算结果的符号位是否一致来进行检测

设Xf,Yf分别为两个操作数的符号位，Sf为结果的符号位，V为溢出标志位，V=1时即表示溢出，

那么就有逻辑表达式：

IMG_256

这个逻辑表达式表明，有符号加法运算溢出的条件是：两个操作数都是正数结果却为负数，或者 两个运算数都是负数结果却是正数。

3) 实验结果的记录与分析：

输入输出见32位快速加法器图

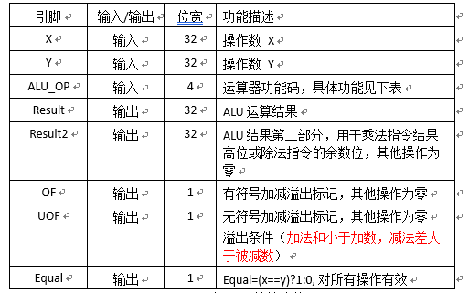
分析 ：将输入数据分为高十六位和低十六位，分别送入2个十六位快速加法器中进行计算。由2个操作数的符号位和运算结果的符号位可计算出是否溢出。

4) 操作步骤及顺序：

* 输入数据分为高十六位和低十六位，分别送入2个十六位快速加法器中
* 将2个十六位快速加法器的P\*,G\*送入CLA74182中，并得出S0和S1
* 由由2个操作数的符号位和运算结果的符号位计算是否溢出
* 由S0和S1推出运算结果S

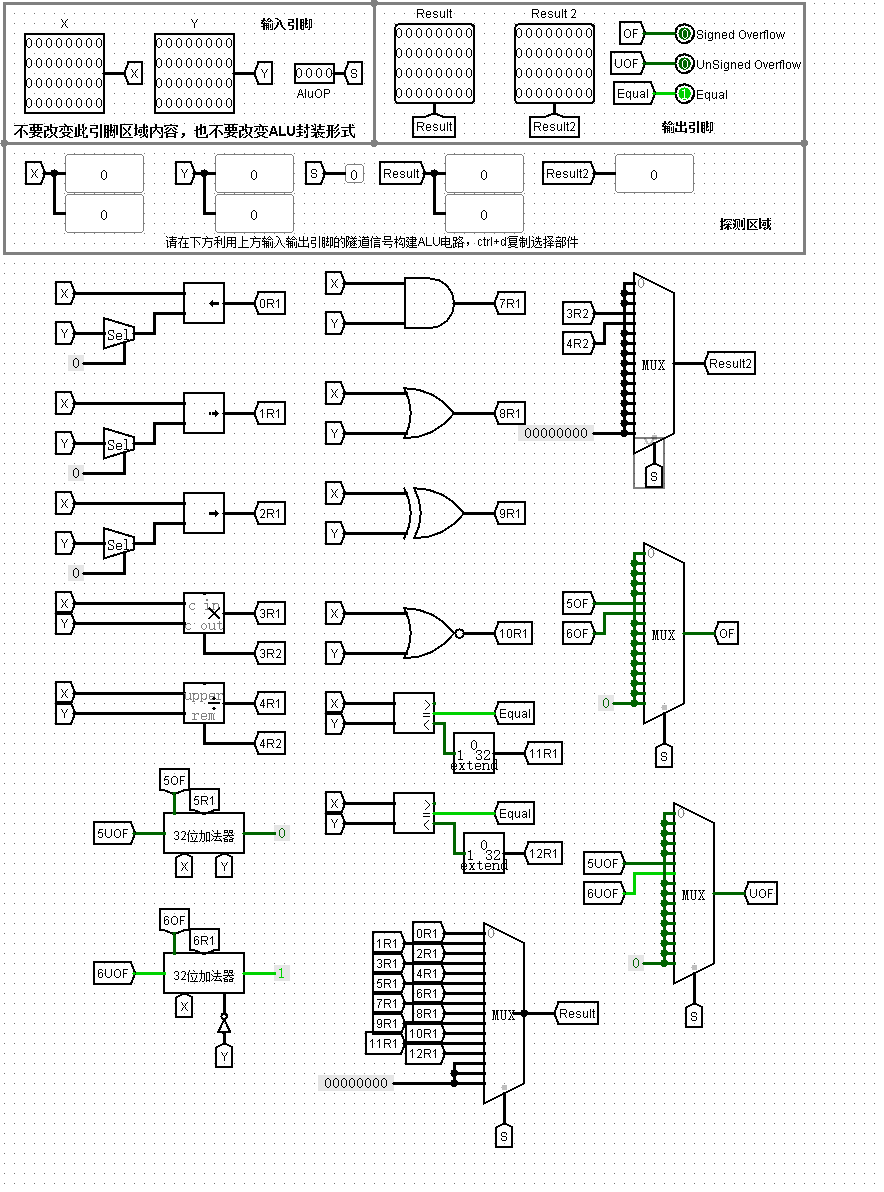
2.6 32位MIPS运算器

**0) 芯片引脚与功能描述.**





**1) 电路图**



**2) 设计分析与说明：**

* 逻辑右移、逻辑左移、算术右移：X表示操作数，Y表示移动的位数
* 乘法运算：R1低32位，R2高32位
* 除法运算：R1为商，R2为余数
* 加减法运算（0加1减）：运算结果存在R1，R2 = 0，区分有符号溢出和无符号溢出
  + 无符号数溢出判断：当最高为向更高位有进位（或借位）时产生溢出。
  + 有符号数溢出判断：最高数据位的进位与符号位的进位位是否一致进行检测
* 按位与，按位或，按位异或，按位或非使用相应器件完成运算
* 符号比较
* 无符号比较

输入：32位操作数1X，32位操作数2Y，AluOP运算器功能控制码S。

输出：结果1Result，结果2Result2，有符号运算溢出判断OF，无符号数溢出判断UOF，两操作数是否相等判断Equal

**3) 实验结果的记录与分析：**

AluOP控制运算符功能，实现相应的运算。（功能较多，不一一举例）

分析：结合各功能的实际原理，分析结果是否正确，验证运算

4) 操作步骤及顺序：

* 输入操作数，根据运算符功能表选择AluOP的对应功能
* 输出运算结果

# 3 存储系统综合实验

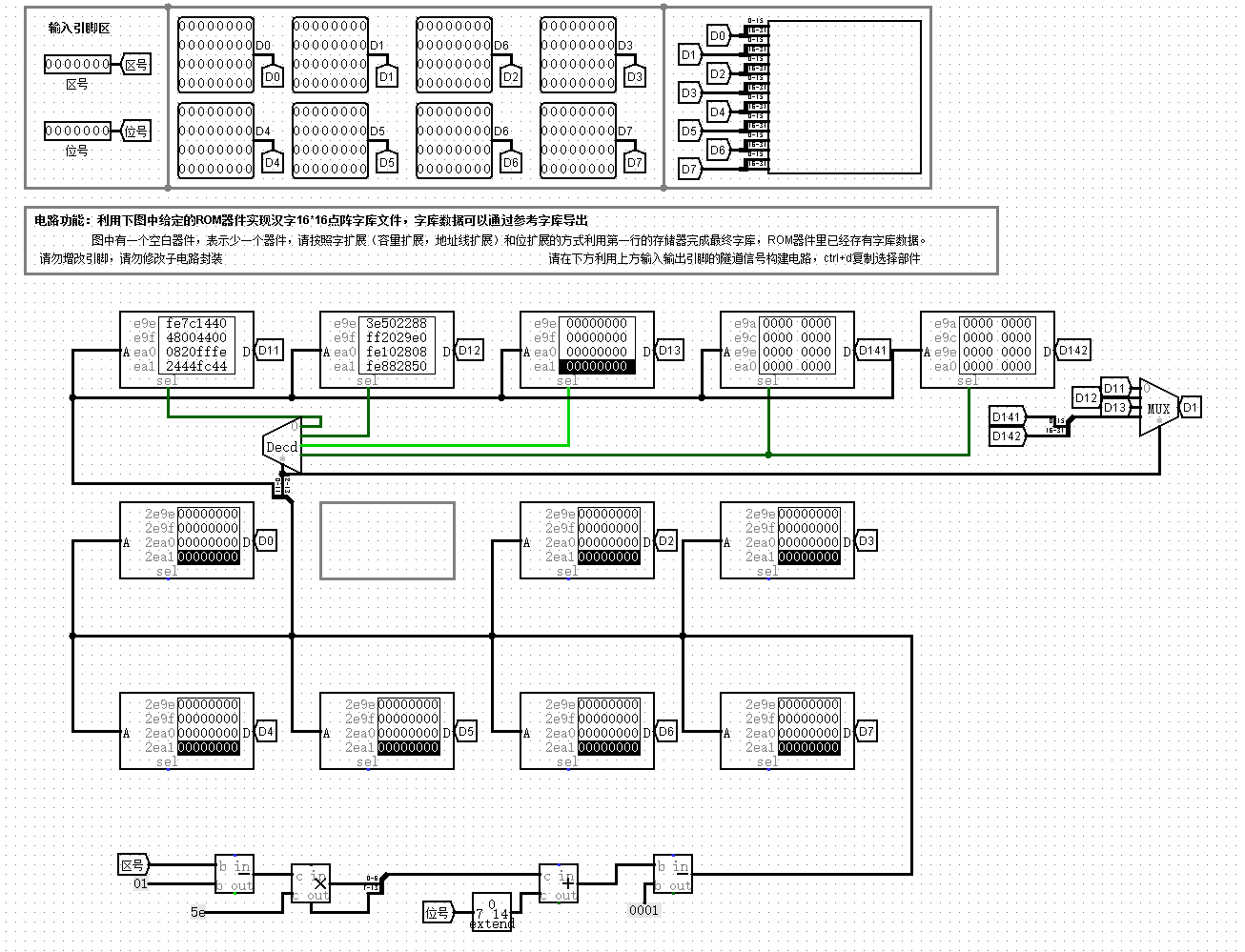
* 1. 存储扩展实验

**实验目的：掌握存储扩展基本原理。**

**实验内容：设计字库文件，利用指定规格存储器进行存储器字扩展。**

**实验要求：现有如下 ROM 部件，2个4K\*16位ROM，3个4K\*32 位ROM，7 个16K\*32 位 ROM，请构建GB2312 1616 点阵字库存储器电路，电路输入为汉字区号和位号，由于16\*16 点阵的字模码需要256 位点阵信息才能显示一个汉字，所以电路输出为 8\*32 位**

**1) 电路图**



2) 设计分析与说明

* 对区号和位号经过一系列处理之后计算出存储器的14位偏移地址
* 其中0-11位位地址线，12和13位为片选信号
* 2位片选信号用来决定选择4个4K\*32中的哪一个
* 12位地址线用来选择相应ROM的地址
* 将选中地址内的数据输出到相应的数据线
* 8个输出引脚D0-D7，每个引脚输出32位数据，总计输出8\*32=256位数据，从而可以表示16\*16字形点阵所需要的256位信息

输入：汉字在GB2312编码中的区号和位号。

注意：这里区号引脚和位号引脚都只接受7个位数据，这是因为GB2312-80将汉字分为94个区和94个位，故7个位（可表示0-127）即可涵盖区间[0,94]。以“边”字为例，其GB2312编码为1763，17表示区号，63表示位号，故为了显示“边”，我们需要往区号引脚输入0010001（17），往位号引脚输入0111111（63）。

输出：GB2312区位码对应的汉字字形码。

电路总共有8个输出引脚D0-D7，每个引脚输出32位数据，总计输出8\*32=256位数据，从而可以表示16\*16字形点阵所需要的256位信息。

**3) 实验结果的记录与分析**

经测试，标准库与待测字库的显示一致，字库实现正确

**4) 列出操作步骤及顺序**

* 对区号和位号经过一系列处理之后计算出存储器的偏移地址，每个偏移地址对应一个16\*16汉字点阵码
* 每次使用同一个存储器偏移地址同时对8个32位ROM进行寻址，单次寻址就可获得8\*32=256位数据，其中每个32位ROM保存16\*16点阵码的固定两行（表示一行需要16位，两行即32位）
* 将2个 4K\*16 ROM位扩展成1个 4K\*32 ROM（使用splitter）
* 将4个 4K\*32 ROM字扩展成1个 16K\*32 ROM（使用multiplexer）
* 字库电路的8个输出引脚（D0-D7）连接至16\*16 LED点阵屏上，从而显示汉字的形状
  1. MIPS寄存器文件设计

**实验目的：为MIPS CPU构造核心功能部件，进一步熟悉多路选择器，译码器，解复用器等Logisim部件的使用**

**实验内容：设计完成满足如下规格要求的 MIPS 通用寄存器组。**

1. 利用 logisim 平台构建一个 MIPS 寄存器组，内部包含 32 个 32 位寄存器，其具体功能如下，具体封装文件为 regfile.circ.

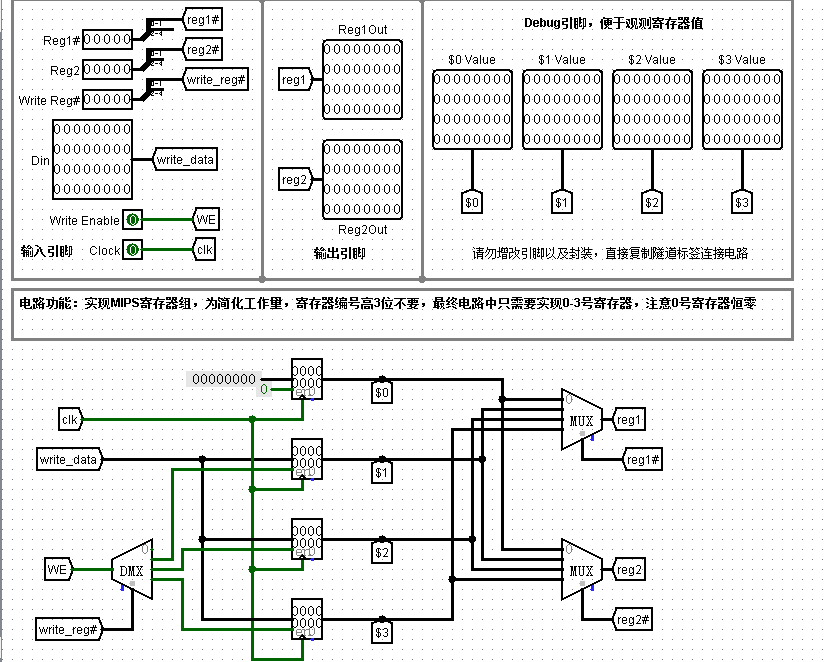
|  |  |  |  |
| --- | --- | --- | --- |
| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| R1# | 输入 | 5 | 读寄存器 1 编号 |
| R2# | 输入 | 5 | 读寄存器 2 编号 |
| W# | 输入 | 5 | 写入寄存器编号 |
| Din | 输入 | 32 | 写入数据 |
| WE | 输入 | 1 | 写入使能信号，为 1 时，CLK 上跳沿将  Din 数据写入 W#寄存器 |
| CLK | 输入 | 1 | 时钟信号，上跳沿有效 |
| R1 | 输出 | 32 | R1#寄存器的值 |
| R2 | 输出 | 32 | R2#寄存器的值 |
| $s0 | 输出 | 32 | 编号为 16 的寄存器的值 |
| $s1 | 输出 | 32 | 编号为 17 的寄存器的值 |
| $s2 | 输出 | 32 | 编号为 18 的寄存器的值 |
| $ra | 输出 | 32 | 编号为 31 的寄存器的值 |

1. 为减少实验中画图工作量，实验工程文件中对 5 位寄存器地址进行了简化，具体见引脚示意图，最终只需实现 4 个寄存器，0 号寄存器功能仍然是恒零。后续实验中如需要使用 32 个寄存器的 MIPS 寄存器文件组，将提供标准组件。
2. 注意时钟信号和电平信号不要混连，时钟仅仅触发状态改变。

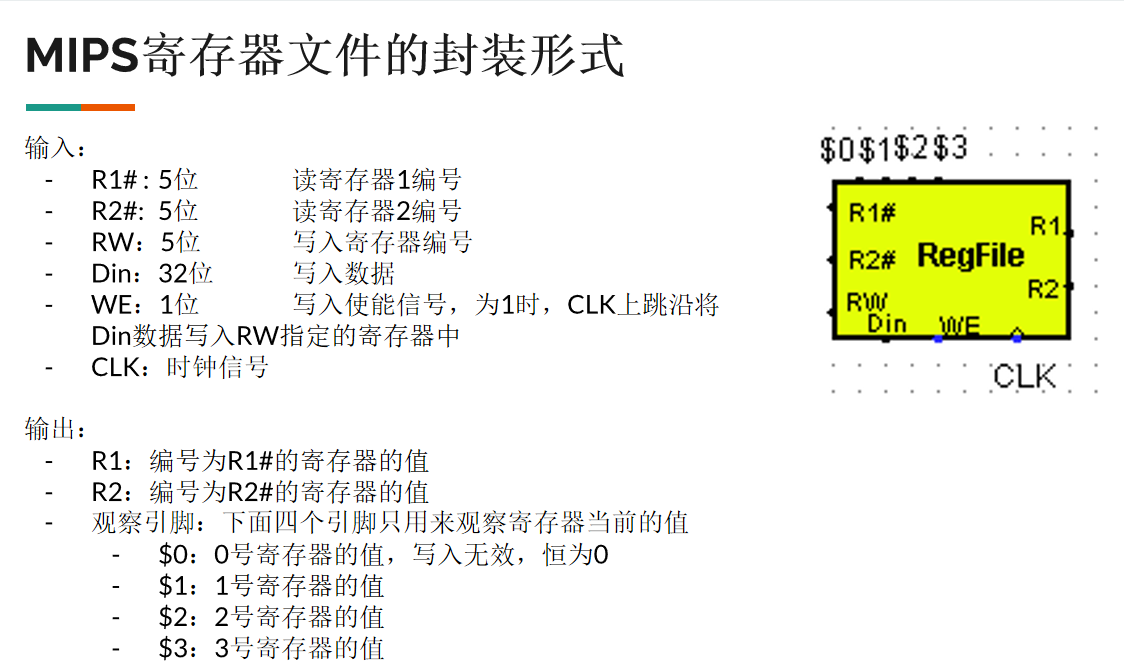
**实验要求：**

* 利用logisim平台构建一个MIPS寄存器组，内部包含32个32位寄存器（实际4个）
* 为减少实验中画图工作量，实验工程文件中对5位寄存器地址进行了简化，具体见引脚示意图，最终只需实现4个寄存器（0-3号），0号寄存器恒为0
* 注意时钟信号和电平信号不要混连，时钟仅仅触发状态改变

**1) 电路图**



**2) 设计分析与说明**



**3) 列出操作步骤及顺序**

输入引脚：

* 分别通过reg1# reg2# write\_reg#获得输入的三个寄存器编号，因为实验只要求实现0-3号寄存器，所以只需提取5位编号输入的低2位（00/01/10/11）
* 通过 write\_data 获取要写入寄存器的32位数据
* 通过 WE 获取写入标志
* 通过 clk 获取时钟信号

输出引脚：

* 将读寄存器1的值输出到reg1
* 将读寄存器2的值输出到reg2
* 将4个寄存器当前保存的值输出到$0,$1,$2,$3

寄存器选择输出：

* 利用多路复用器（multiplexer），多路复用器可以通过select bits选择多个输入中的一个进行输出
* 将四个寄存器的输出引脚同时接到多路复用器的输入引脚上，将reg1#和reg2#作为多路复用器的select bits，从而选择指定输入进行输出。

寄存器选择写入：

* 利用解复用器（demultiplexer），解复用器可以将输入只通过select bits指定的线路输出。
* 将write\_data同时连接到四个寄存器的data引脚（由于0号寄存器恒为0，因此可以忽略data引脚，或者将en置0）
* 将we引脚作为解复用器的输入
* 将write\_reg#作为解复用器的select bits
* 通过解复用器将要写入数据的寄存器的enable引脚置1，下个时钟信号到来时，只有该写入寄存器的enable引脚为1，因此wrtie\_data数据只会写入到该寄存器中（其余寄存器enable引脚为0，时钟信号不会触发寄存器写入），从而实现寄存器选择写入

# 4 心得体会

在构建不同类型的加法器（如串行加减法器、先行进位电路和快速加法器）时，我不仅学会了如何实现它们，还理解了它们的工作原理和优势。

通过存储扩展实验，我了解到了存储器字位拓展的具体实现，掌握了存储扩展基本原理。之前知识会在纸上画画简单的电路图，使用logisim设计电路图之后，更加了解了电路的一些细节设计，掌握了一些器件的用法已经各引脚的功能。

通过MIPS 寄存器文件设计,我进一步熟悉多路选择器，译码器，解复用器等Logisim部件的使用。