華中科技大學

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

1.1 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

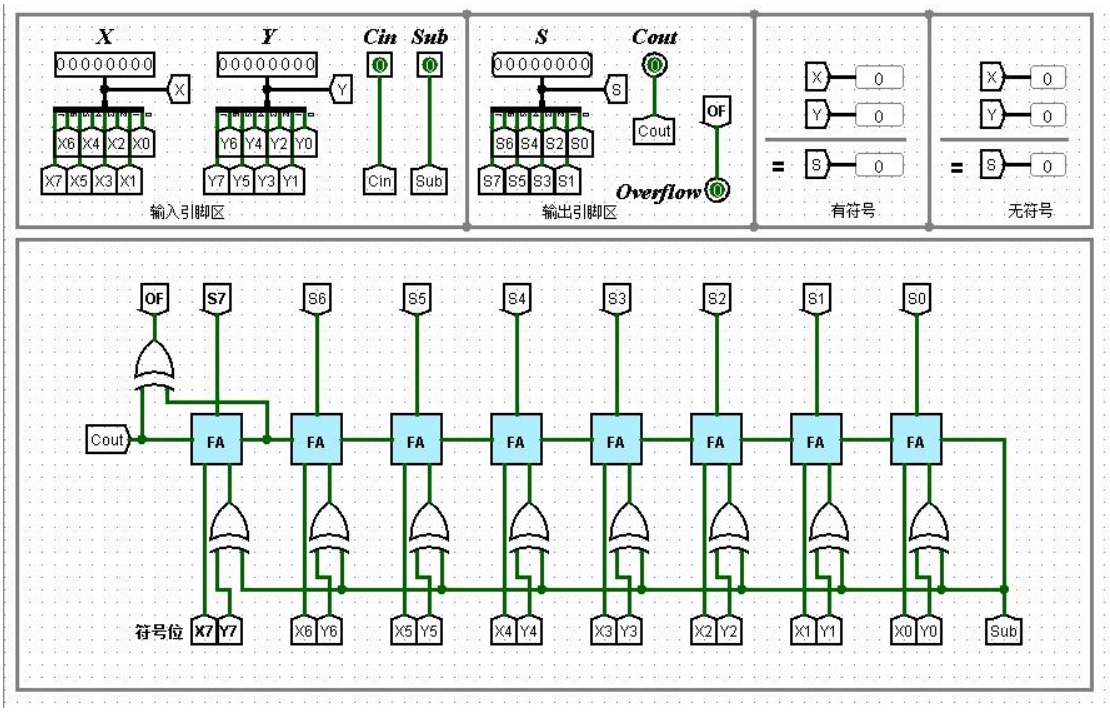
1.3 实验环境

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 ：

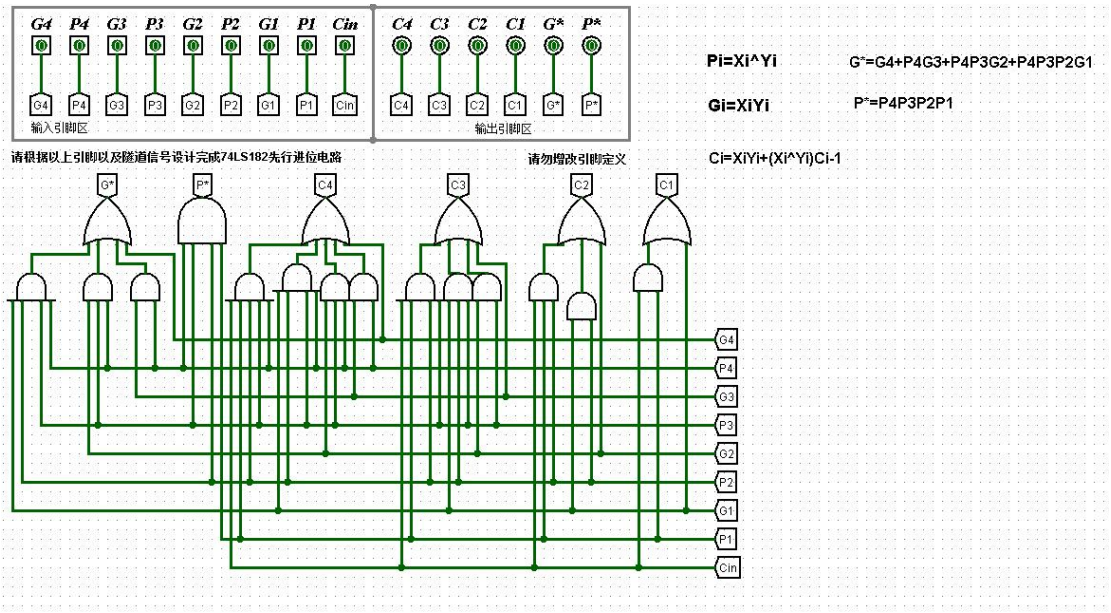
sub = 0 ，两个正数执行 x + y 操作。输出结果为 s=0000 0001，没有发生溢出，最高位的进位为 0。

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：

基本算术逻辑运算单元

2、串行进位和并行进位

（2）并行进位

$$C_1 = A_0B_0 + (B_0 + A_0)C_0$$

$$C_2 = A_1B_1 + (B_1 + A_1)C_1$$
$$= A_1B_1 + (A_1 + B_1)A_0B_0 + (A_1 + B_1)(A_0 + B_0)C_0$$

$$C_3 = A_2B_2 + (B_2 + A_2)C_2$$
$$= A_2B_2 + (B_2 + A_2)(A_1B_1 + (B_1 + A_1)A_0B_0 + (A_1 + B_1)(A_0 + B_0)C_0)$$
$$= A_2B_2 + (A_2 + B_2)(A_1B_1 + (A_1 + B_1)A_0B_0) + (A_2 + B_2)(A_1 + B_1)(A_0 + B_0)C_0$$

$$C_4 = A_3B_3 + (B_3 + A_3)C_3$$
$$= A_3B_3 + (A_3 + B_3)(A_2B_2 + (A_2 + B_2)(A_1B_1 + (A_1 + B_1)A_0B_0))$$
$$+ (B_3 + A_3)(B_2 + A_2)(A_1 + B_1)(A_0 + B_0)C_0$$

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

4. 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 ：

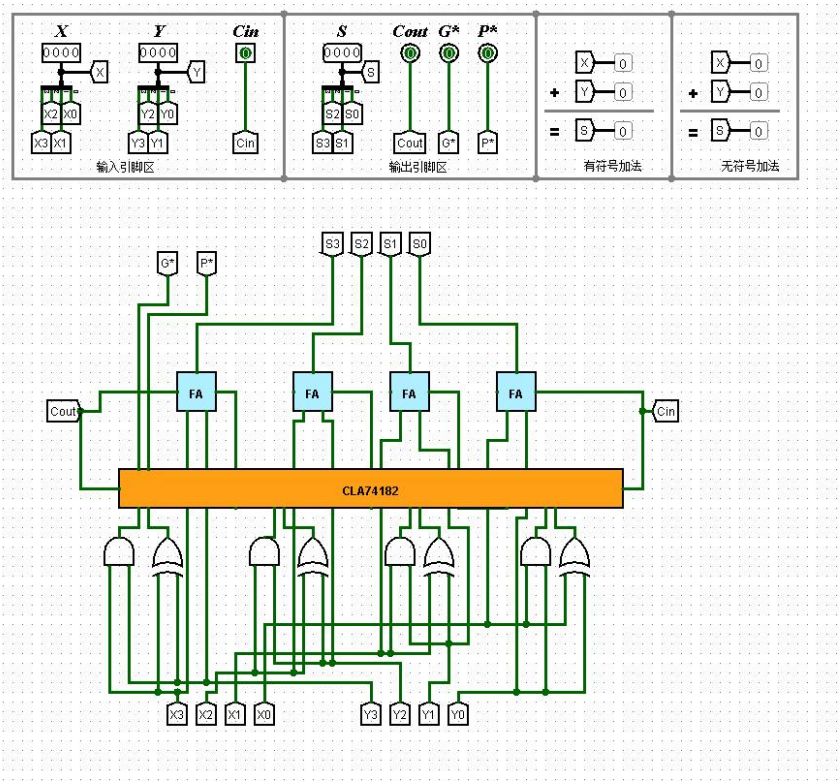
C1，C2，C3, C4 的计算不需要彼此依赖，而是可以独立地根据 A3-A0，B3-B0 以及 C0 计算出来。

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

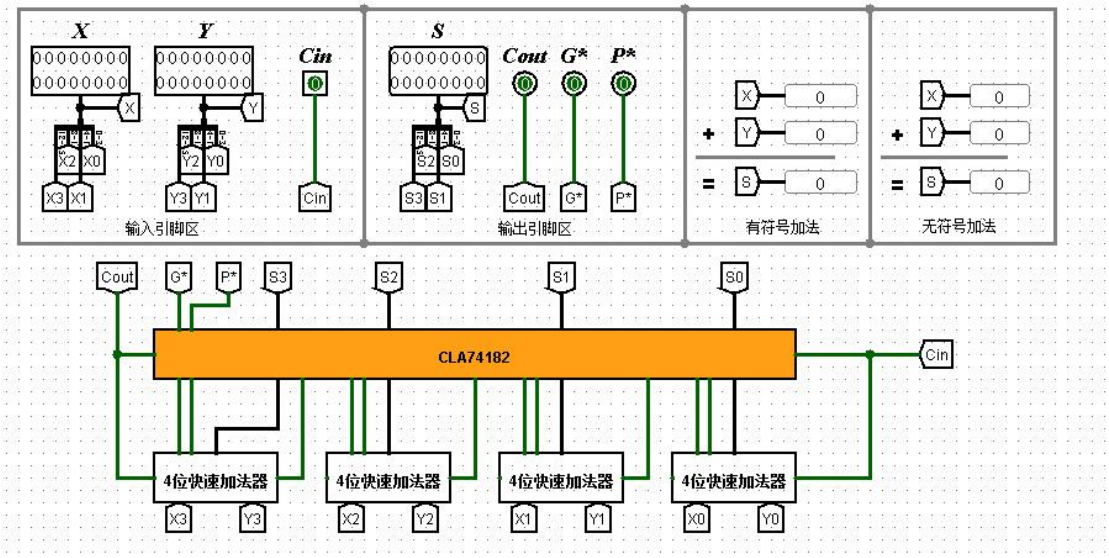
分析：先对 Xn、Yn 分别进行 AND 和 OR 运算，送入 CLA74182，再由此算出 S3-S0

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。

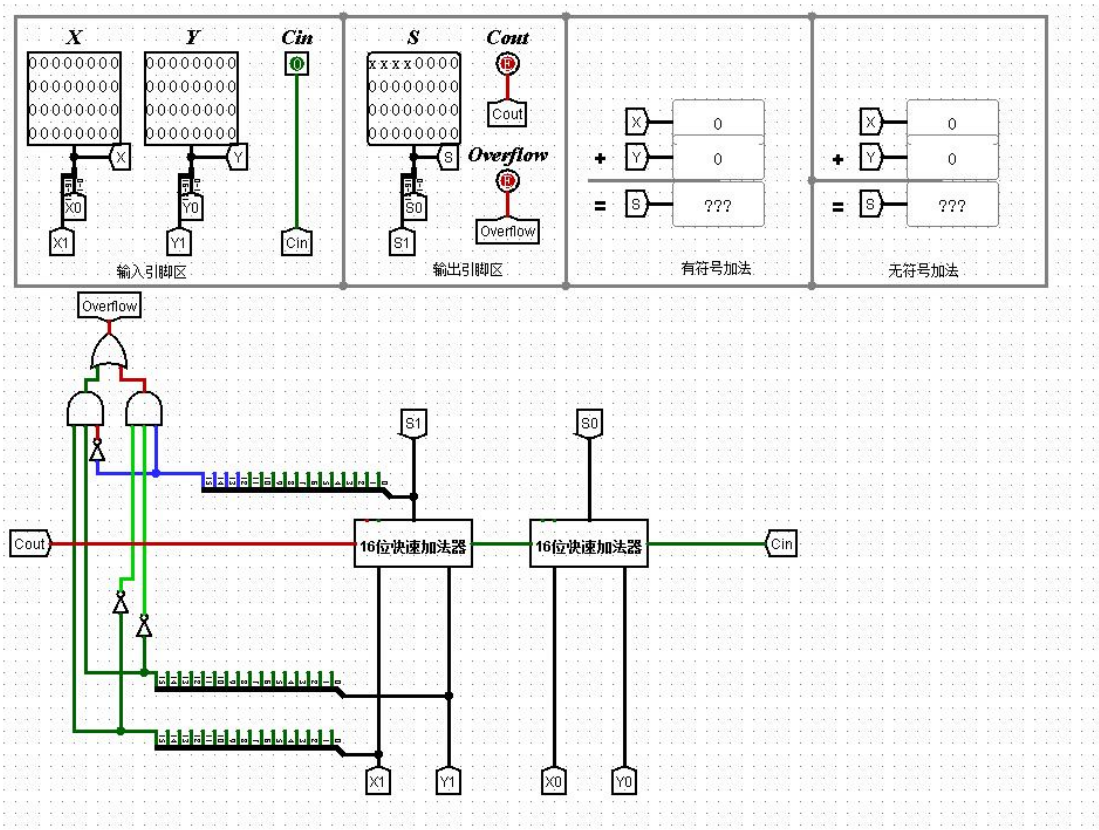
4) 操作步骤及顺序：

- 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：

$$V = X_f Y_f \overline{S}_f + \overline{X}_f \overline{Y}_f S_f$$

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

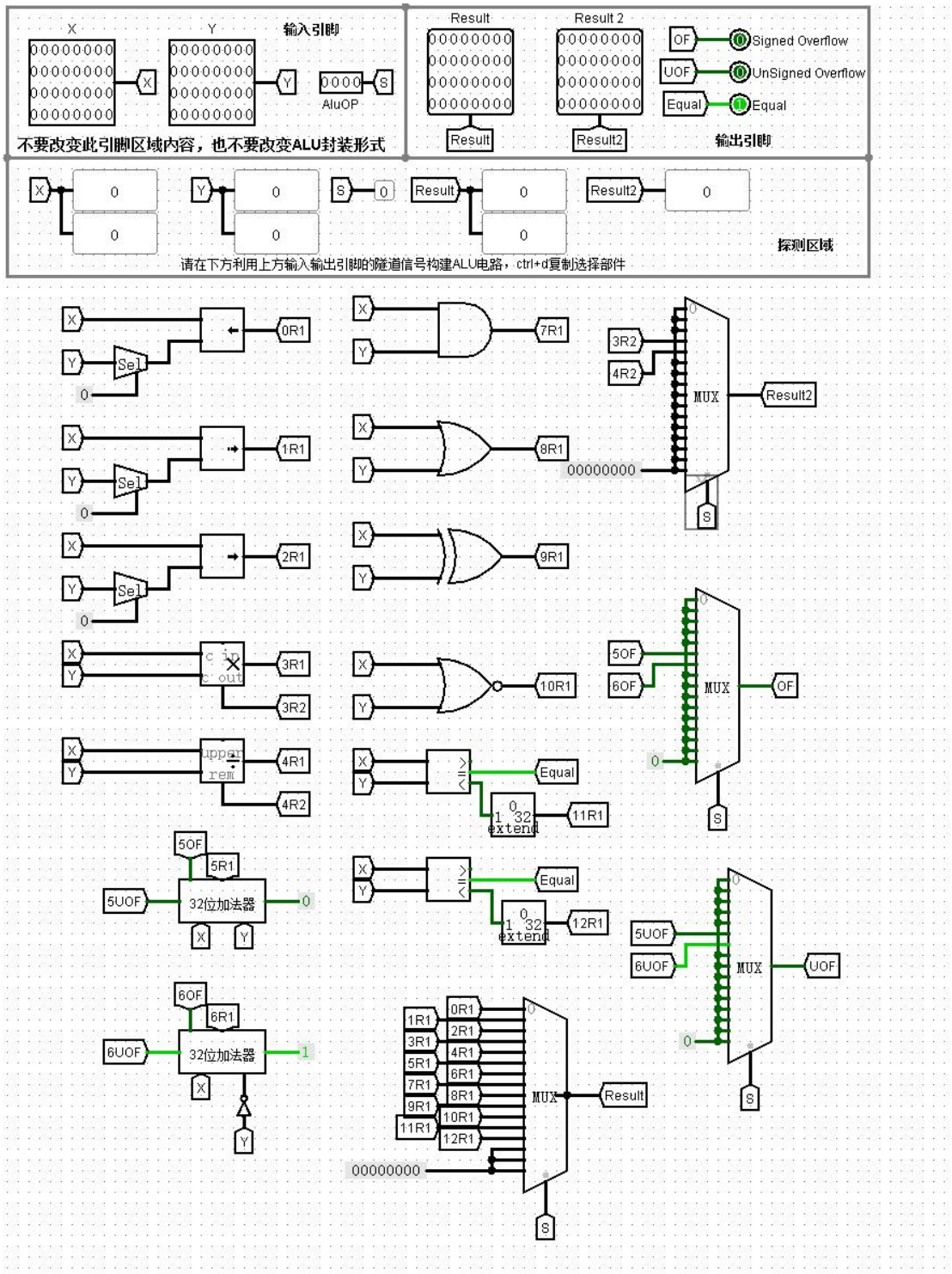
2.6 32 位 MIPS 运算器

0) 芯片引脚与功能描述.

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见下表
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零 溢出条件（加法和小于加数，减法差大于被减数）
Equal	输出	1	Equal=(X==Y)?1:0, 对所有操作有效

ALU OP	十进制	运算功能
0000	0	Result = X << Y 逻辑左移 （Y取低五位） Result2=0
0001	1	Result = X >>>Y 算术右移 （Y取低五位） Result2=0
0010	2	Result = X >> Y 逻辑右移 （Y取低五位） Result2=0
0011	3	Result = (X * Y)[31:0]; Result2 = (X * Y)[63:32] 有符号
0100	4	Result = X/Y; Result2 = X%Y 无符号
0101	5	Result = X + Y (Set OF/UOF)
0110	6	Result = X - Y (Set OF/UOF)
0111	7	Result = X & Y 按位与
1000	8	Result = X | Y 按位或
1001	9	Result = X⊕Y 按位异或
1010	10	Result = ~(X |Y) 按位或非
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

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

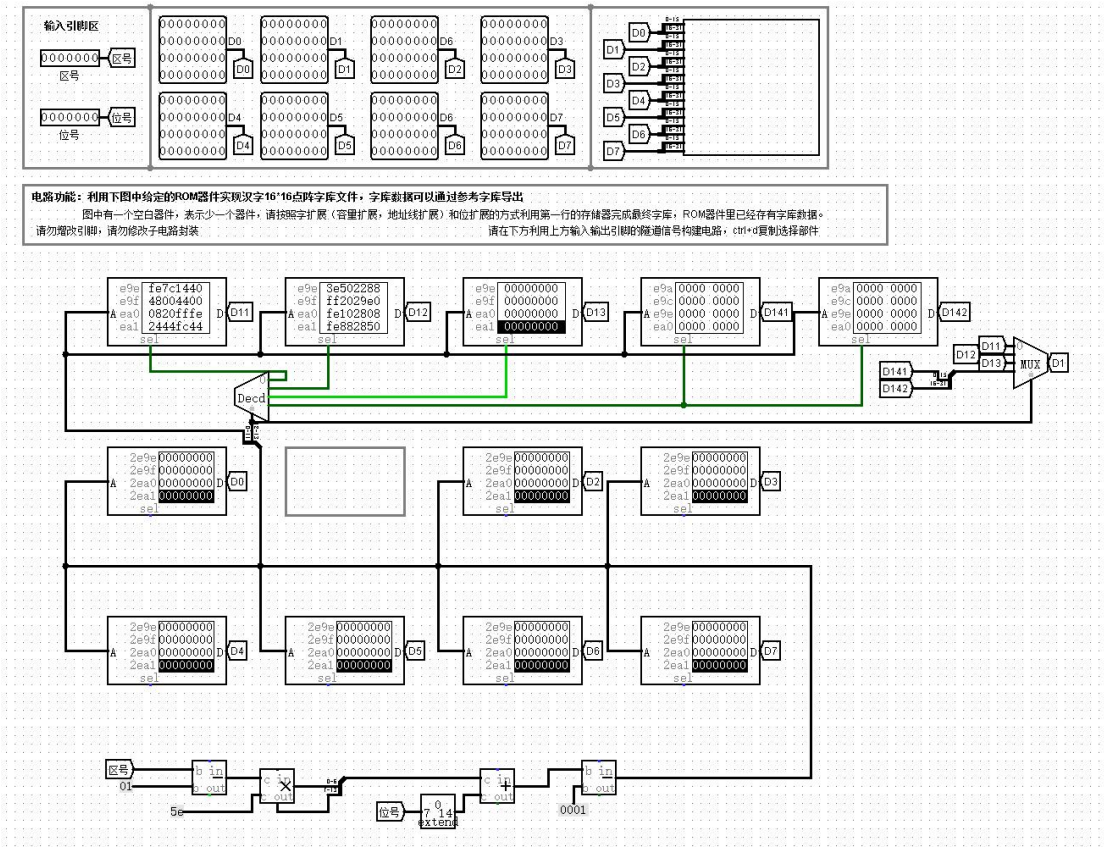
3.1 存储扩展实验

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

3.2 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

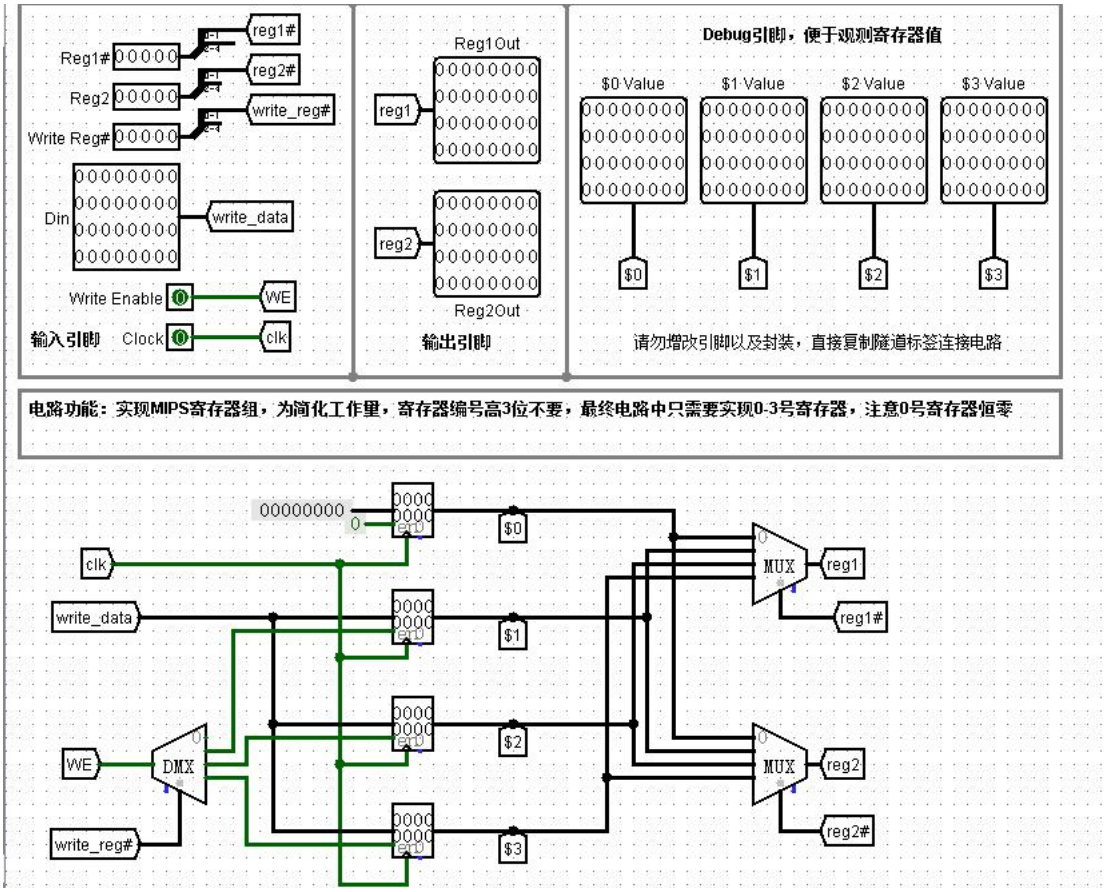
2) 为减少实验中画图工作量，实验工程文件中对 5 位寄存器地址进行了简化，具体见引脚示意图，最终只需实现 4 个寄存器，0 号寄存器功能仍然是恒零。后续实验中如需要使用 32 个寄存器的 MIPS 寄存器文件组，将提供标准组件。

3) 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

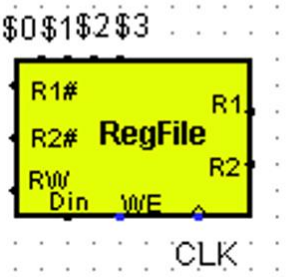


2) 设计分析与说明

MIPS寄存器文件的封装形式

输入：

- R1#：5位　　读寄存器1编号
- R2#：5位　　读寄存器2编号
- RW：5位　　写入寄存器编号
- Din：32位　　写入数据
- WE：1位　　写入使能信号，为1时，CLK上跳沿将Din数据写入RW指定的寄存器中
- CLK：时钟信号



输出：

- R1：编号为R1#的寄存器的值
- R2：编号为R2#的寄存器的值
- 观察引脚：下面四个引脚只用来观察寄存器当前的值
  - $0：0号寄存器的值，写入无效，恒为0
  - $1：1号寄存器的值
  - $2：2号寄存器的值
  - $3：3号寄存器的值

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。