

# 华中科技大学

## 2016

### 计算机组成原理

### ·实验报告·

专    业：                计算机科学与技术

班    级：                CS1409

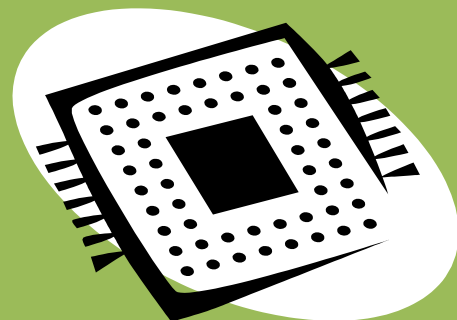
学    号：                U201414795

姓    名：                王卓焱

电    话：                18986274247

邮    件：                513905392@qq.com

完成日期：                2017-01-13



计算机科学与技术学院

# 华中科技大学课程实验报告

---

## 目 录

<b>1</b>	<b>运算器实验.....</b>	<b>2</b>
1.1	设计要求 .....	2
1.2	方案设计 .....	4
1.3	实验步骤 .....	10
1.4	故障与调试 .....	11
1.5	测试与分析 .....	14
<b>2</b>	<b>存储器实验.....</b>	<b>22</b>
2.1	设计要求 .....	22
2.2	方案设计 .....	23
2.3	实验步骤 .....	24
2.4	故障与调试 .....	26
2.5	测试与分析 .....	29
<b>3</b>	<b>CPU 实验.....</b>	<b>33</b>
3.1	设计要求 .....	33
3.2	方案设计 .....	35
3.3	实验步骤 .....	43
3.4	故障与调试 .....	49
3.5	测试与分析 .....	54
<b>4</b>	<b>总结与心得.....</b>	<b>60</b>
4.1	实验总结 .....	60
4.2	实验心得 .....	60
	<b>参考文献.....</b>	<b>62</b>

## 1 运算器实验

### 1.1 设计要求

#### 1.1.1 快速加法器设计

利用基本逻辑门电路构造 4 位具有先行进位特征的快速加法器，并进行子电路封装。利用封装好的 4 位快速加法器构建 32 位组内先行进位，组间先行进位的加法器，并分析对应电路延迟。

#### 1.1.2 运算器封装实验

利用 logisim 平台中现有运算部件构建一个 32 位运算器，可支持算数加、减、乘、除，逻辑与、或、非、异或运算、逻辑左移、逻辑右移，算术右移运算，支持常用程序状态标志（有符号溢出 OF、无符号溢出 CF，结果相等 Equal），运算器功能以及输入输出引脚见下表，在主电路中详细测试自己封装的运算器。下表格 1-1 是芯片和功能描述，表格 1-2 是运算符功能表

表格 1-1 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
Equal	输出	1	$Equal=(x==y)?1:0$ ，对所有操作有效

# 华中科技大学课程实验报告

表格 1-2 运算符功能

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

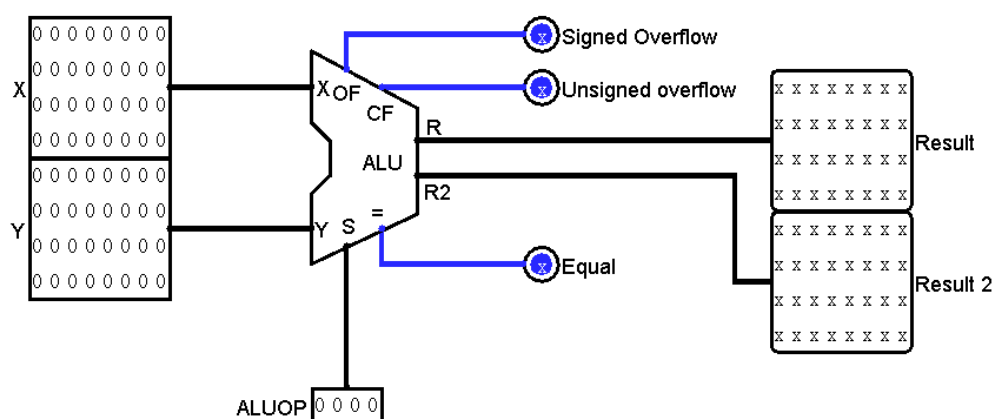


图 1-1 运算器封装示意图

## 1.2 方案设计

### 1.2.1 四位并行加法器的设计

四位并行加法器的设计时将各个位的进位通过迭代的方式，迭代成关于输入信号量的而不是中间变量的表达式，其中存在的并行加法进位链如下：

$$C_1 = X_1 Y_1 + (X_1 \oplus Y_1) C_0 = G_1 + P_1 C_0$$

$$C_2 = X_2 Y_2 + (X_2 \oplus Y_2) C_1 = G_2 + P_2 C_1$$

$$= G_2 + P_2 (G_1 + P_1 C_0)$$

$$= G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = X_3 Y_3 + (X_3 \oplus Y_3) C_2 = G_3 + P_3 C_2$$

$$= G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 C_0)$$

$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = X_4 Y_4 + (X_4 \oplus Y_4) C_3 = G_4 + P_4 C_3$$

$$= G_4 + P_4 (G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0)$$

所以在进行计算之前，需要将  $P_1$  -4 和  $G_1$  -4 进行计算，P 的作用是当参加运算的数取某些值使得改为为 1 时，则说明进位由下端的输入进位产生，G 的作用是说明进位输出是由参加运算的两个数直接产生，它们的存在使得设计更多位的加法器成为了可能。

P, G 的接法如下图 1-2:

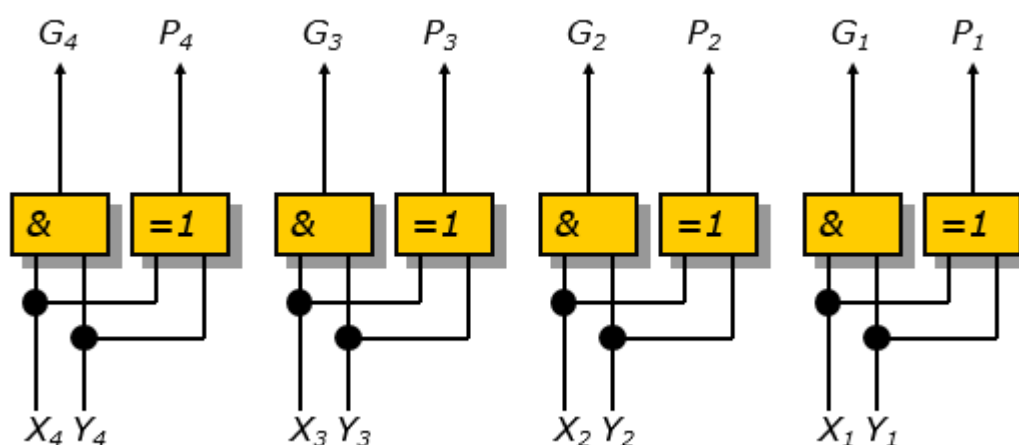
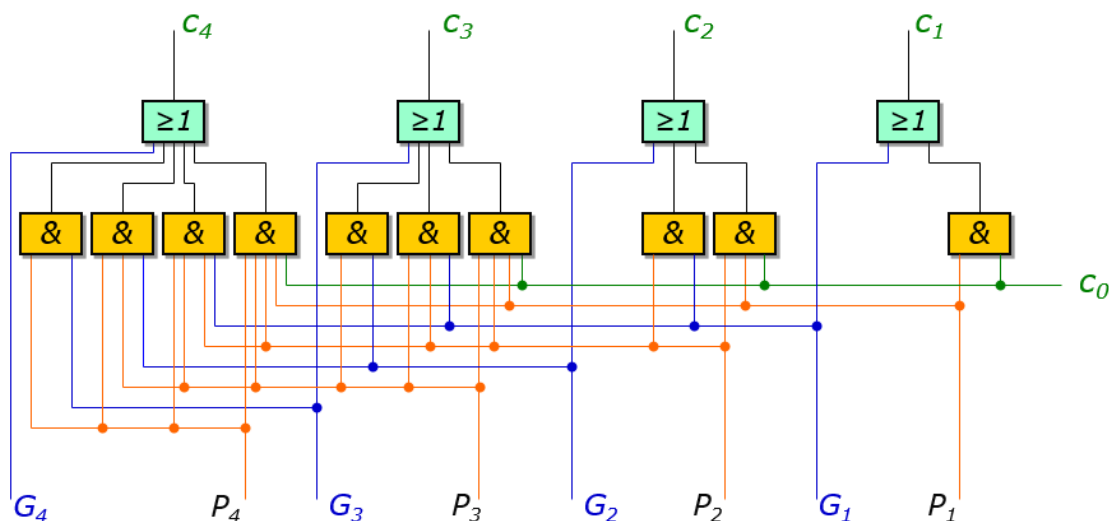


图 1-2 GP 连接方式图

可以看出，该电路有一级的延迟。

然后根据输出的 **P** 和 **G** 信号，就可以对四位并行加法器进行连接。

四位先行进位加法器电路是对输入的四个 **PG** 信号进行运算，而不是对 **X** 和 **Y** 进行直接运算，根据先行进位的关系可得图 1-3:



从上图可以看出，进位信息就是通过 PG 来确定的，PG 又是通过对 X 和 Y（输入信号）的引用确定的，所以最终的进位实际也是来自于 X 和 Y 的并行关系来确定的。

在得到了先行进位的进位信息后，就可以使用这些信息（ $C_{0-4}$ ）来计算最终的结果。最终四位的计算结果也是通过  $P$  与进位信号  $C$  来确定的，四位的从低到高的排列拼接起来就是四位先行进位快速加法器的最终输出，根据  $X$  和  $Y$  的关系可以得出  $S$  和  $GP$  的关系确定如下：

- $S_4=X_4 \oplus Y_4 \oplus C_3=P_4 \oplus C_3$
- $S_3=X_3 \oplus Y_3 \oplus C_2=P_3 \oplus C_2$
- $S_2=X_2 \oplus Y_2 \oplus C_1=P_2 \oplus C_1$
- $S_1=X_1 \oplus Y_1 \oplus C_0=P_1 \oplus C_0$

如果有多位的话，再进行四位的芯片组合，即可实现多位先行进位的快速加法器的设计。

# 华中科技大学课程实验报告

确定了输出结果后，就可以根据上述的是两个电路的过程确定出四位并行加法器的设计图 1-4:

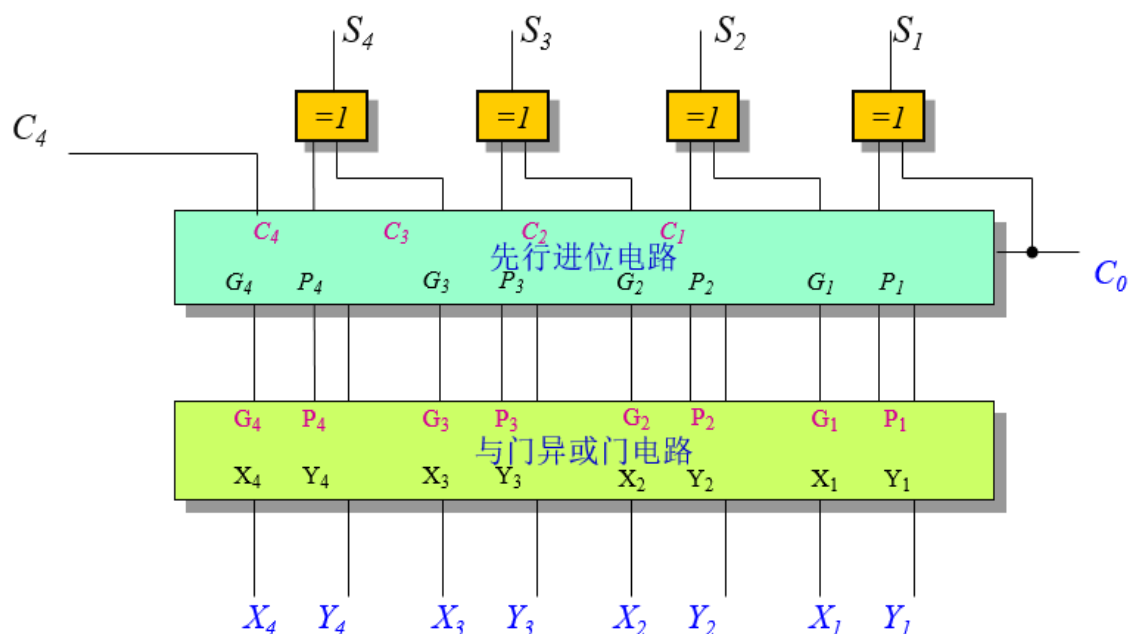


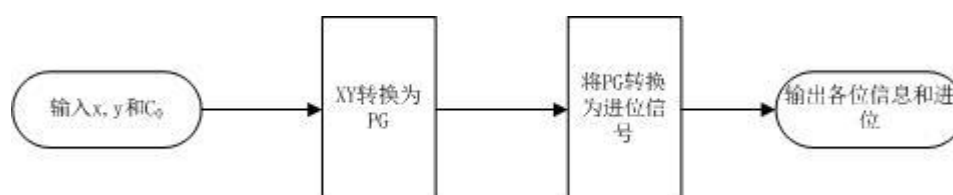
图 1-4 四位并行加法器设计图

可以分析出，该图的电路延迟等级是四级。

该电路包括输入四位的  $X$  和  $Y$ ，输入进位位  $C_0$ ，输出有四位输出结果  $S$  和进位输出  $C_4$ ，中间使用了两个芯片： $X$  和  $Y$  到  $P$  和  $G$  的转换电路、根据  $P$  和  $G$  确定进位位的先行进位电路，然后在主电路中根据进位信息即可确定出输出结果的各个位的信息。

在该电路设计成功后，四级先行进位加法器就设计完毕，将该思路在 logisim 布线设计后即可。

设计方式位先设计与与异或门电路，再设计先行进位电路，然后将两部分连接起来，最终确定出输出端口，连线测试。



## 1.2.2 32 位加法器的构造

32 位加法器的构造是建立在 4 位并行加法器的基础上的,32 为加法器的也设计为组内和组间都并行的方式,因为在四位加法器中组内输出了  $P^*$  和  $G^*$ ,所以对位数的扩展,只需要进行几个四位加法器的并行连接即可,每一组的  $P^*$  和  $G^*$  来自最后的输出判断。

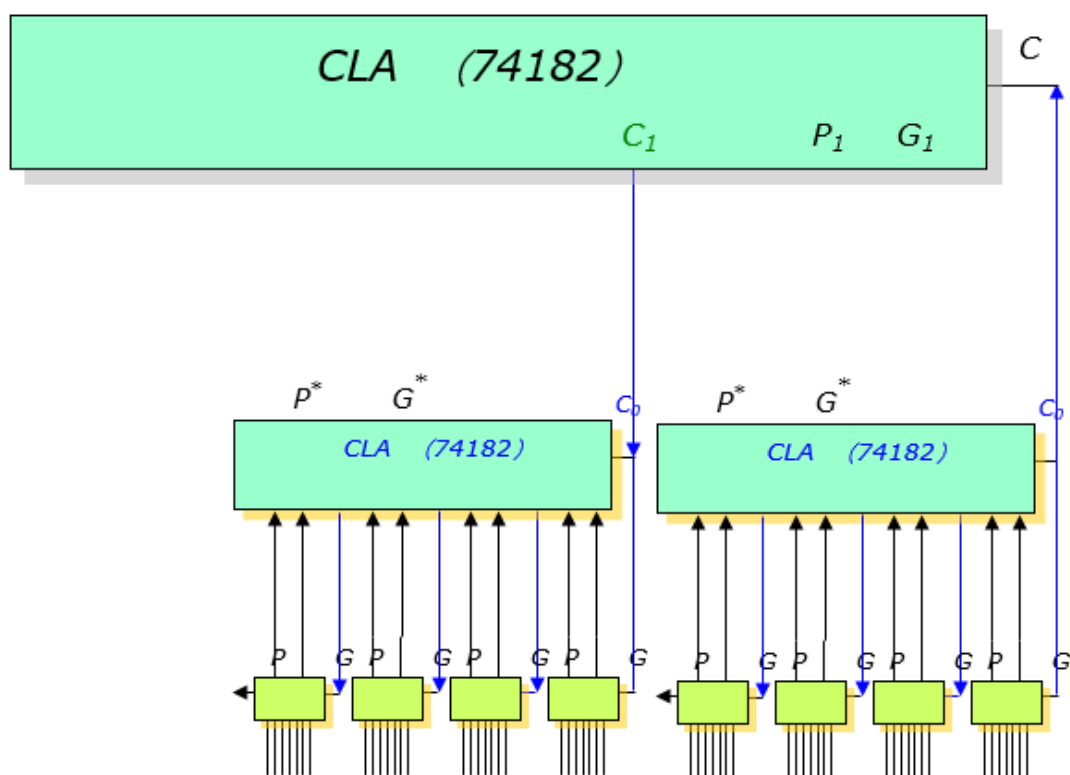


图 1-5 32 位先行进位电路连接方式

对图 1-5 的解析:

在最下边的是四位并行加法器,每位都输出了  $P^*$  和  $G^*$ ,把相邻的  $P^*$  和  $G^*$  送入 74182 电路中,得出后位的进位信息,将填满的 74182 再送入更高一级的 74182 芯片中,就可以得出输出信息和各组之间的进位关系,根据进位关系,就可以确定下一级的输入数据。

下边 8 个 4 位并行加法器,共 32 位,从高到低拼合,就得到了 32 位运算后的结果。

该电路的基础上,再补充好先对 32 位运算后的溢出检测,整个电路便可以完成。



溢出检测方式:

溢出检测的方法分为有符号溢出和无符号溢出两种。

- 有符号溢出检测 OF:

有符号溢出检测在此处使用的是对最高数据位的进位与符号位的进位位是否一致进行检测。

当运算过程中最高数据位的进位与符号位进位位不同步产生时就表明运算结果产生了溢出，即：

$$OF = C_D \oplus C_F$$

在有符号的检测中，加法和减法的判断方式是一样的。

- 无符号溢出检测 UOF

无符号检测较为简单，只需要判断最高位的进位位就可以判断是否溢出，但是对于加法和减法，判断条件是不同的。

对于加法，判断进位位是否为 1，如果为 1 则代表溢出。

对于减法，判断进位位是否为 0，如果为 0 则代表溢出。（因为减法实际是取补码进行的加法运算，所以结果是 0 才代表溢出）

根据上述，对 32 位并行加法器加入溢出检测，即可在 logisim 中将该设计进行。

## 1.2.3 运算器封装实验

运算器封装实验是在我们的 32 位加法器的基础上，进行一定量的功能增加，然后将这些功能通过操作码进行区分，不同的操作码对应不同的功能（加减乘除左移右移等等）。

实验中需要注意的就是功能选择实现的方法，分为两种，第一种是通过选择运算功能，然后在输入数据后，在功能确定后，才可以调用相应功能；第二种是在输入数据后，就进行了所有的运算器操作，在输出时候，通过操作码的不同使用多路选择器进行输出选择。第一种方法的优点是运算少，占用资源较少，但是缺点是时间周期太长，如果数据来了再来操作码的话，就会存在一个等待的过程；第二种方法的优点是等待周期少，输入后进行并行计算，同时对操作码进行选择，从而在一定程度上体现了并行性，但是缺点是在此情况下的存在中间存储变量信息过大的情况。

# 华中科技大学课程实验报告

在实验中选择了第二种方法，即同时计算多个情况，然后进行选择输出，基本流程框图图 1-6:

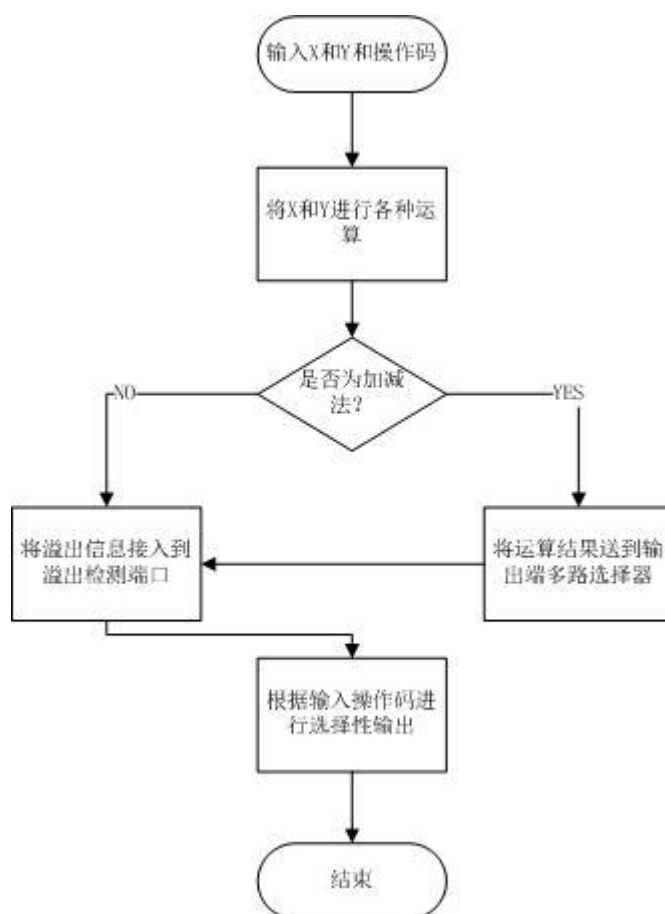


图 1-6 运算器结构图

在图 1-6 中检测溢出只需要检测加法和减法，所以 OF 和 UOF 位只需要连这两个运算的，其余端口置零。输出的 result 有两个，其中 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

$\text{Equal} = (x == y) ? 1 : 0$ ，对所有操作有效。

输出接口 2 (result2) 接口只对乘除法适用，其余端口多路选择器输入端直接置零。

比较器直接调用 logisim 系统库中的比较器，这样虽然简单，但是带来的问题是可能会使电路运行较慢，但是此处电路较为简单，基本观察不到这种情况

在上述基础上进行 logisim 布线，即可完成本次实验。

## 1.3 实验步骤

### 1.3.1 实验准备

- 1) 复习有关运算器的内容，对数据通路的构成、数据在数据通路中的流动及控制方法有基本的了解。
- 2) 熟悉电路中各部分的关系及信号间的逻辑关系
- 3) 设计实验电路，画出各模块的图，注意各引脚的标注，节省实验的时间。

### 1.3.2 实验步骤

- 1) 熟悉开发平台，根据所给要求，先设计出一位的加法器
- 2) 根据公式写出  $P, G$ （由给出的  $X$  和  $Y$  确定）
- 3) 在一位加法器设计成功后，使用并行电路，设计出四位的快速加法器，其中输入有四位的两个加数  $X, Y$ ，还有进位  $C_0$ ，输出有和数  $S$  和先行进位位  $G^*, P^*$ ，先行进位位是用来确定下一位的进位情况，有了先行进位位，就可以进行多个累加器的并发，顺利的完成位数的扩展。
- 4) 在完成 4 位加法器设计后，根据组间并行的规则，利用  $PG$  的传递规则，写出 32 位并行加法器，并且确定溢出检测。
- 5) 在完成加法器后，利用自己的加法器来做有无符号的加减运算。并且利用 logisim 中已经存在的各种运算器（包括乘除，唯一，判断等），构建出 32 位的运算器。
- 6) 在构建出 32 位运算器后，构造其溢出检测，然后准备测试
- 7) 测试通过后，进行 32 位运算器的封装。
- 8) 测试最终结果是否通过。



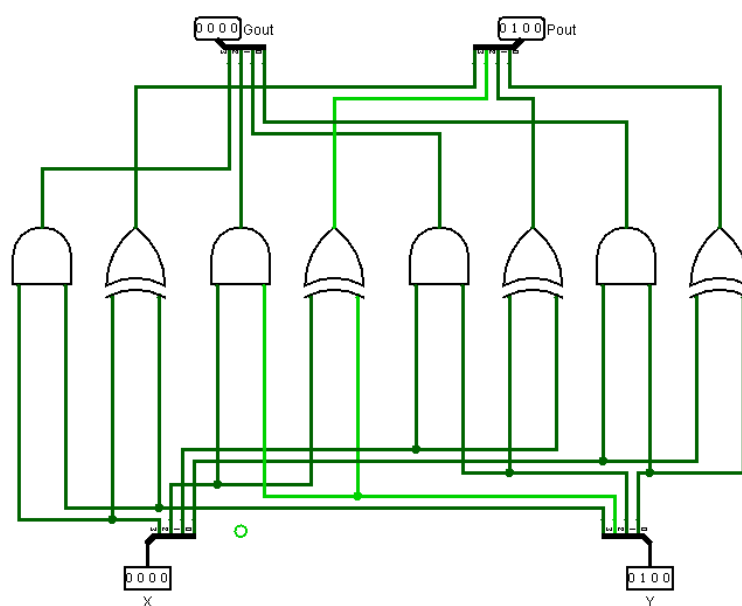


图 1-8 修改后的电路图

## 1.4.2 进位判断出错

**故障现象：**如图 1-9 在进行无符号溢出检测的时候发现加法后减法的溢出检测结果出现了错误。

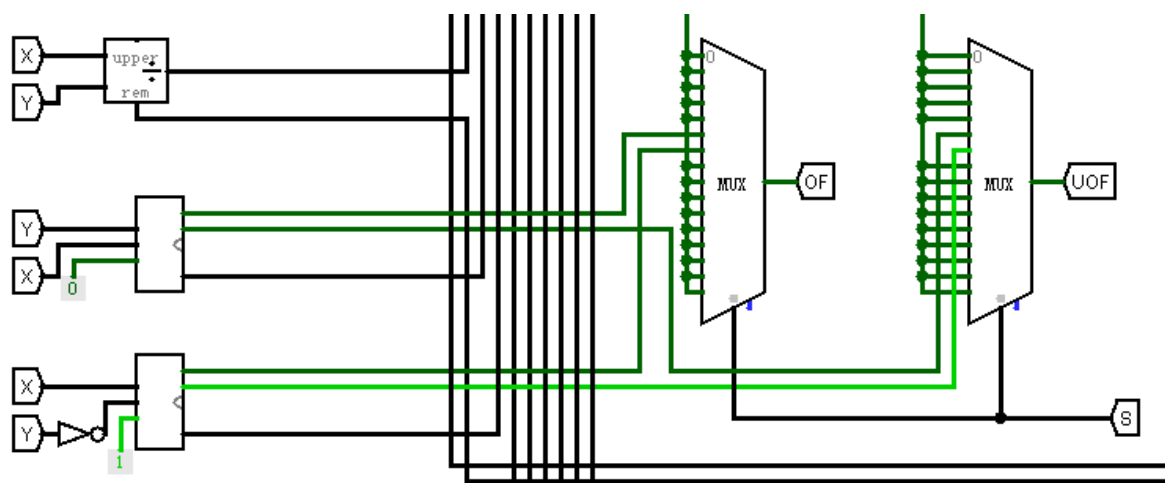


图 1-9 溢出判断错误图

**原因分析：**无符号检测时候，开始以为将进位连出去，如果进位检测是 1 的话就代表溢出，但是在实际测试时候发现，如果是减法的话，实际的判断溢出是 0 而不是 1，因为减法是通过补码加法实现的。

**解决方案：**在减法无符号的溢出检测端口加个反门，如图 1-10。

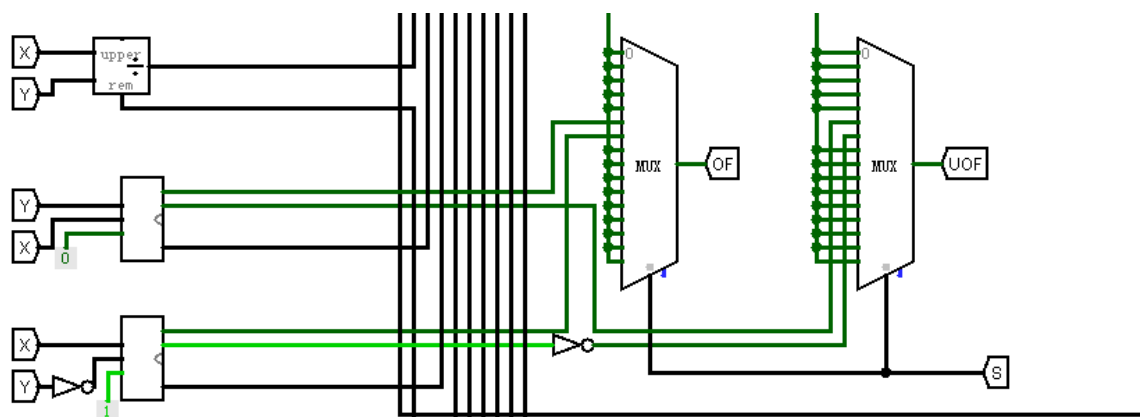


图 1-10 修改后的电路图

## 1.4.3 先行进位设计出错

**故障现象：**在第一次设计时候，根据 PPT 的 32 位加法器进行设计，如图 1-11 最后出错。

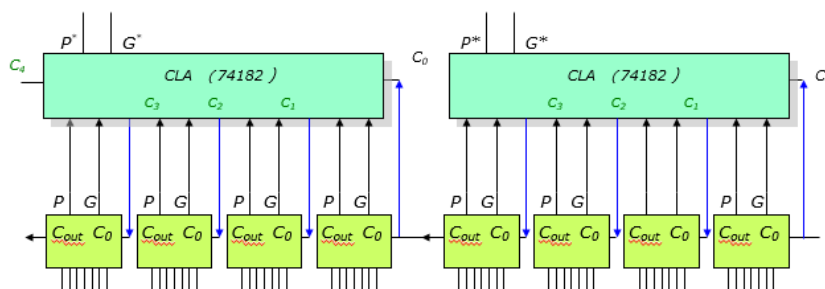


图 1-11 32 位加法器设计图

**原因分析：**实验中需要的是组间也为并行，而此处的组间设计为了串行，明显出错。

**解决方法：**使用正确的电路并行设计图进行解决（在设计中给出图 1-5）。

# 华中科技大学课程实验报告

## 1.5 测试与分析

溢出测试用例见表格 1-3。（为了表中显示方便，所有数字均为无符号十进制数字）

表格 1-3 溢出信号测试用例

#	A	B	OP	运算	有符号溢出	无符号溢出	运算结果 1	运算结果 2
1	1	4	0101	加	○	○	5	0
2	2281701376	2022989824	0101	加	○	●	9723904	0
3	1610612736	1614807040	0101	加	●	○	3225419776	0
4	2147483648	2147483648	0101	加	●	●	0	0
5	16	1	0110	减	○	○	15	0
6	0	1073741825	0110	减	○	●	3221225471	0
7	2147483648	1	0110	减	●	○	2147483647	0
8	1073741824	3221225472	0110	减	●	●	2147473648	0
9	2048	8	0011	乘			16384	0
10	2097152	2097152	0011	乘			0	1024
11	2048	516	0100	除			3	500
12	8	2	0000	逻辑左			32	0
13	8	2	0010	逻辑右			2	0
14	8	2	0001	算术右			2	0
15	9	3	0111	与			1	0
16	9	3	1000	或			11	0
17	9	3	1001	异或			10	0
18	1	7	1010	或非			4294967288	0
19	33	33	1100	无符号比			0	0
20	21	2	1011	有符号比			1	0

对上述功能进行测试演示：

测试电路放入封装好的隧道中，只需要输入数字即可完成运算。

# 华中科技大学课程实验报告

#1: 无溢出加法图 1-12:

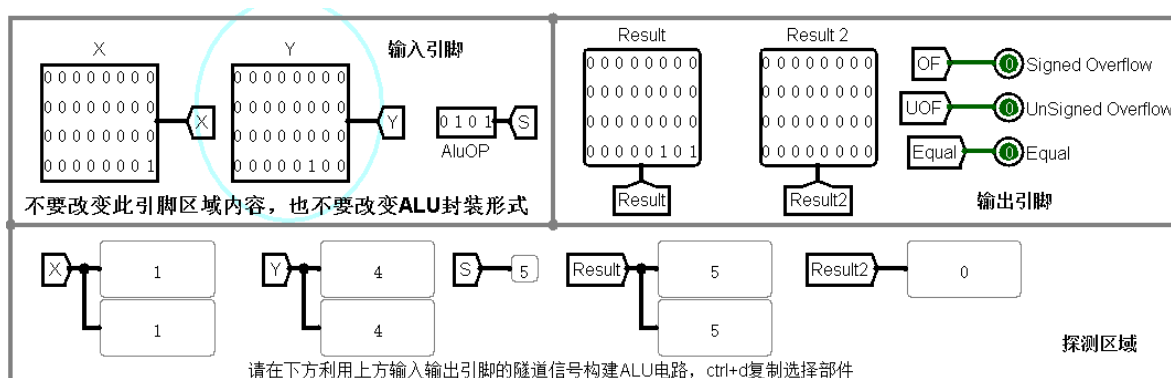


图 1-12 无溢出加法图

#2: 无符号溢出加法图 1-13:

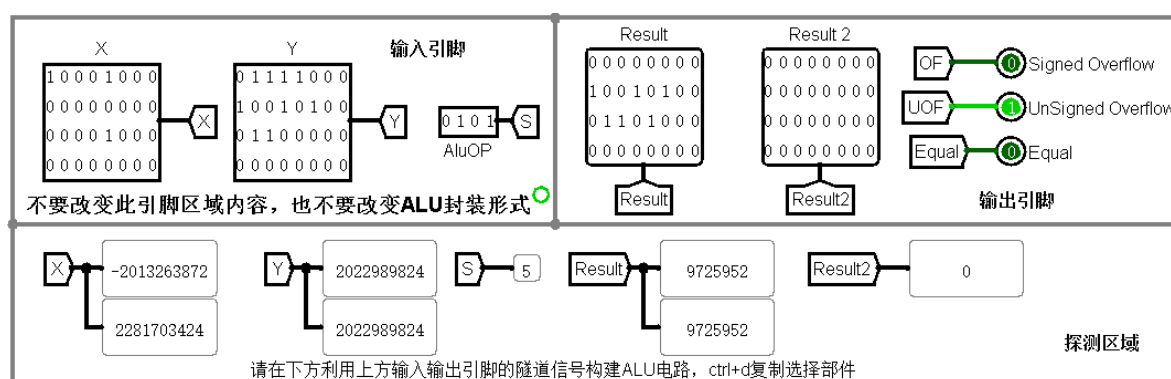


图 1-13 无符号溢出加法图

#3: 有符号溢出加法图 1-14:

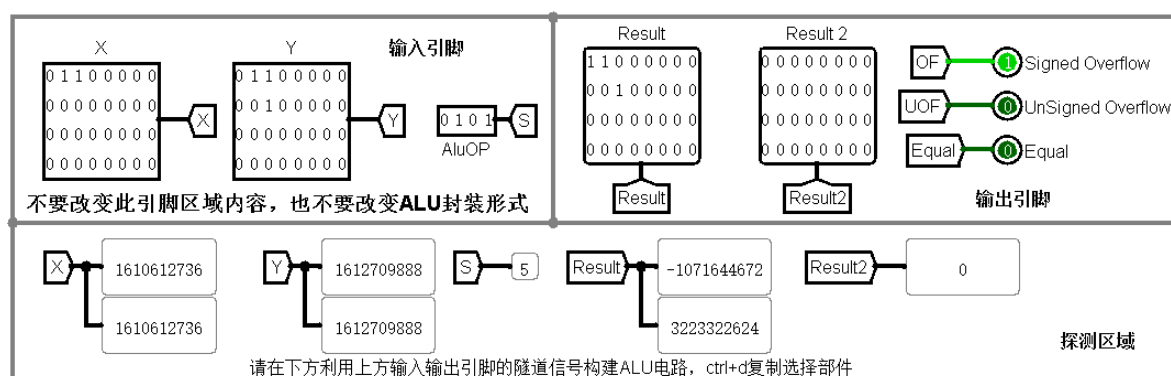


图 1-14 有符号溢出加法图



# 华中科技大学课程实验报告

#4: 有符号无符号同时溢出加法图 1-15:

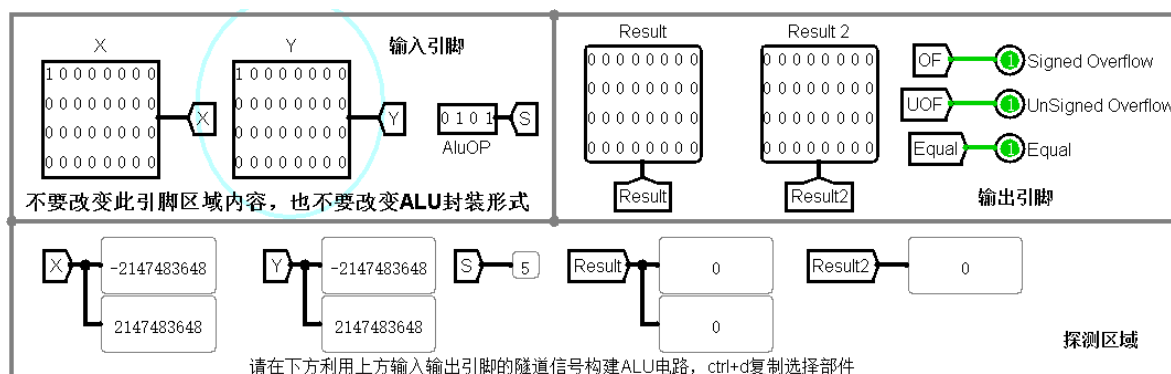


图 1-15 有符号无符号同时溢出加法图

#5: 无溢出减法图 1-16:

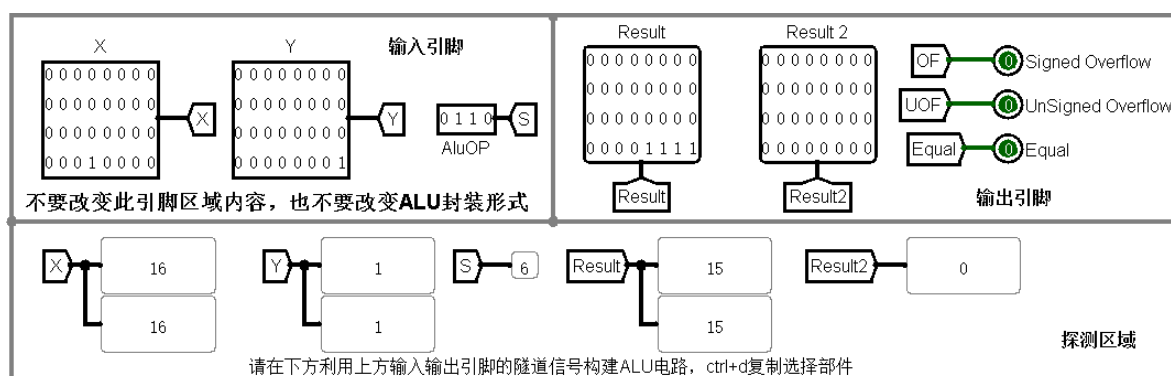


图 1-16 无溢出减法图

#6: 无符号溢出减法图 1-17:

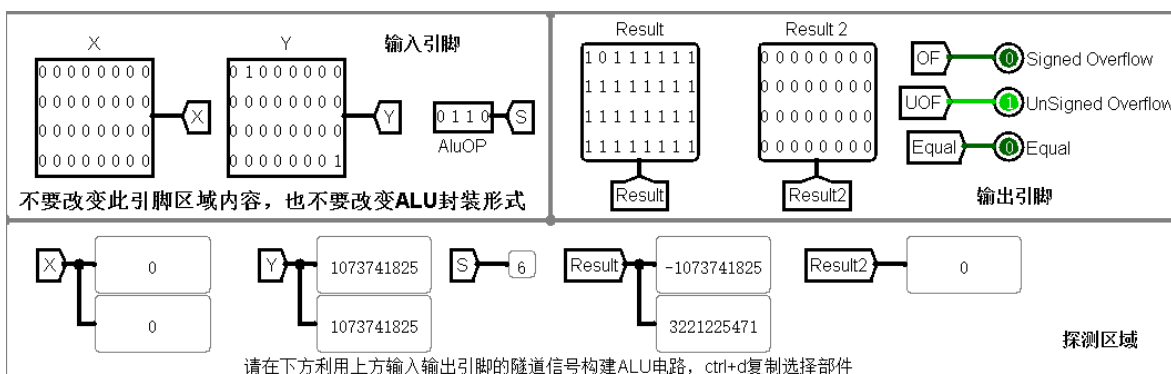


图 1-17 无符号溢出减法图

# 华中科技大学课程实验报告

#7: 有符号溢出减法图 1-18:

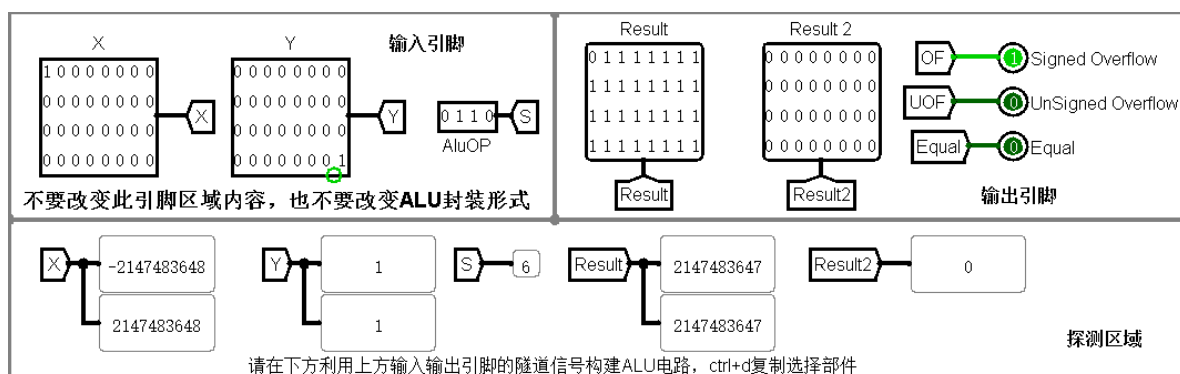


图 1-18 有符号溢出减法图

#8: 溢出减法图 1-19:

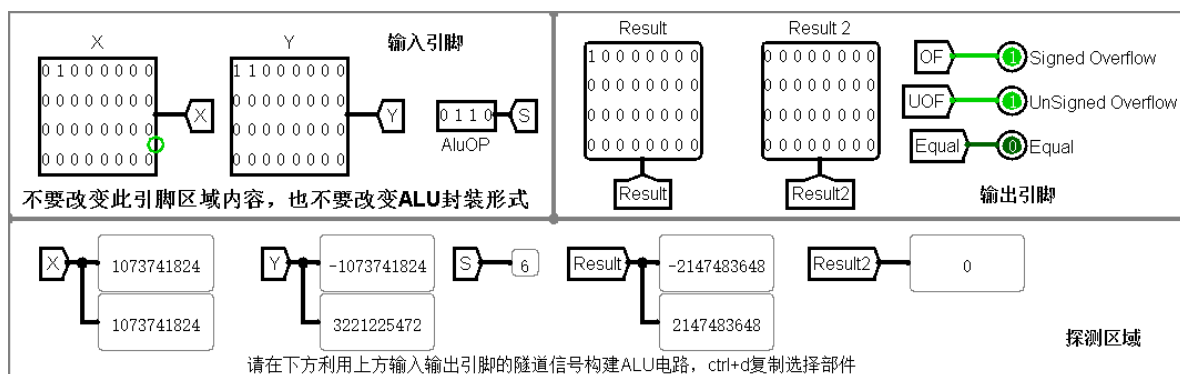


图 1-19 有无符号均溢出减法图

#9: 乘法图 1-20:

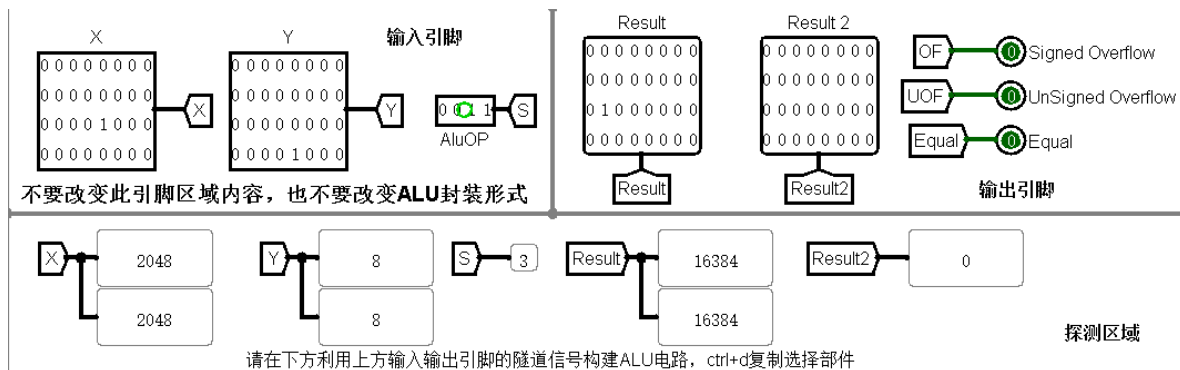


图 1-20 乘法示意图

# 华中科技大学课程实验报告

#10: 用到两个结果位的乘法图 1-21:

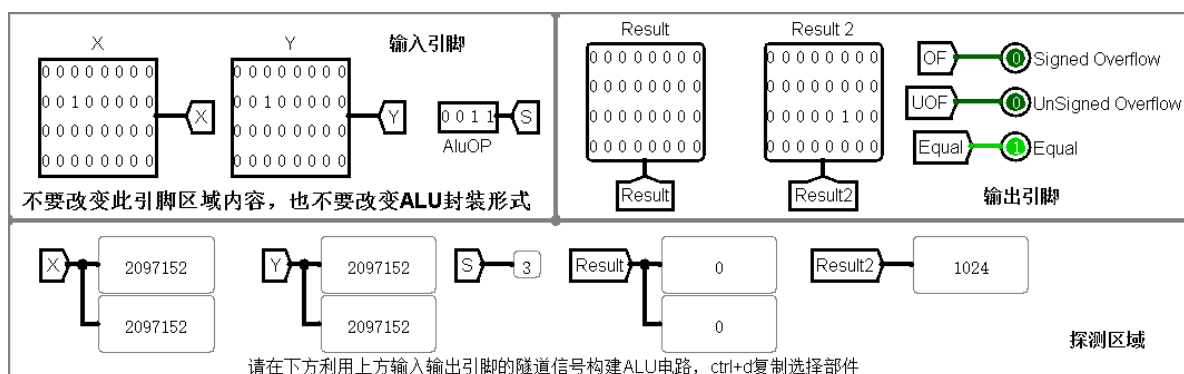


图 1-21 大数乘法示意图

#11: 除法图 1-22:

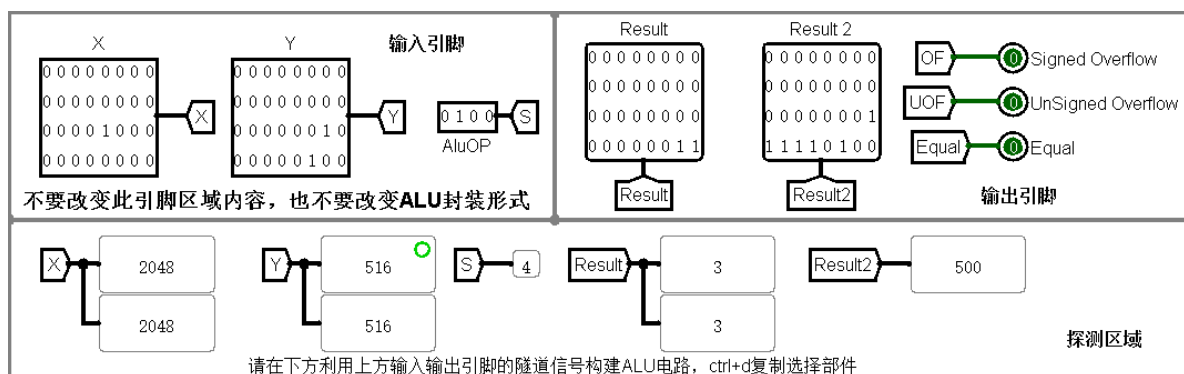


图 1-22 除法示意图

#12: 逻辑左移图 1-23:

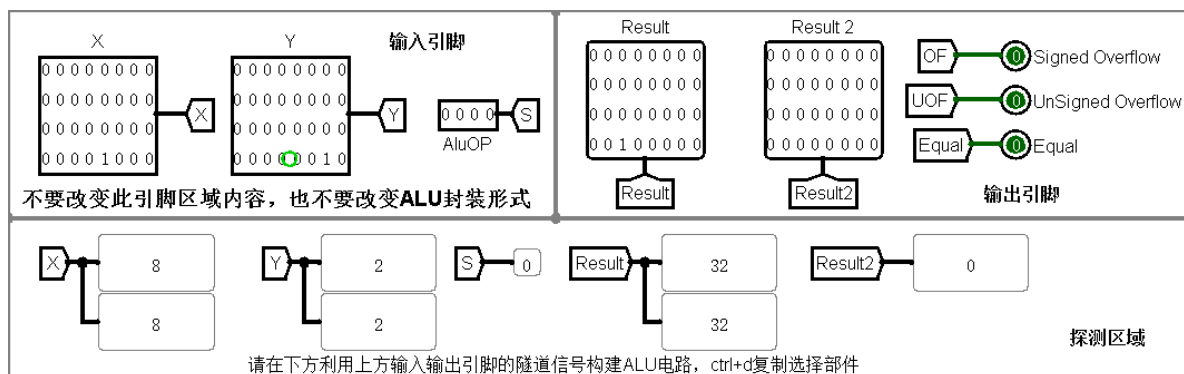


图 1-23 逻辑左移示意图

# 华中科技大学课程实验报告

#13: 逻辑右移图 1-24:

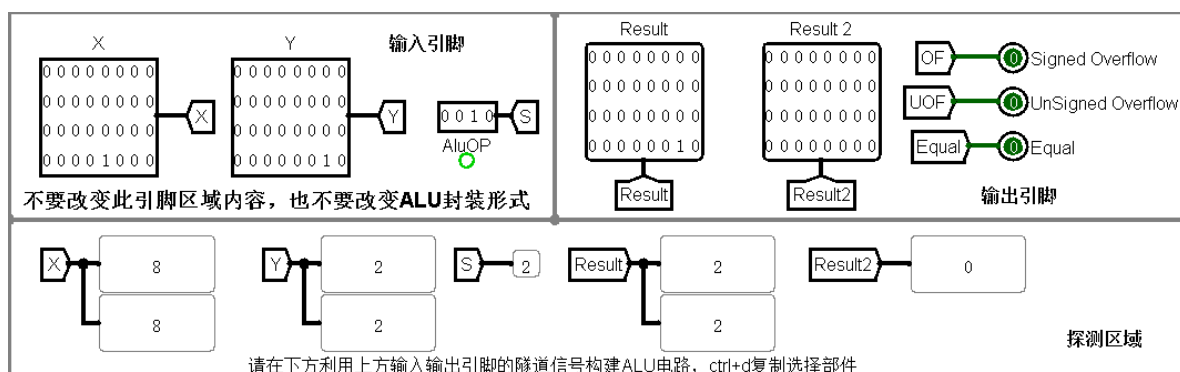


图 1-24 逻辑右移示意图

#14: 算术右移图 1-25:

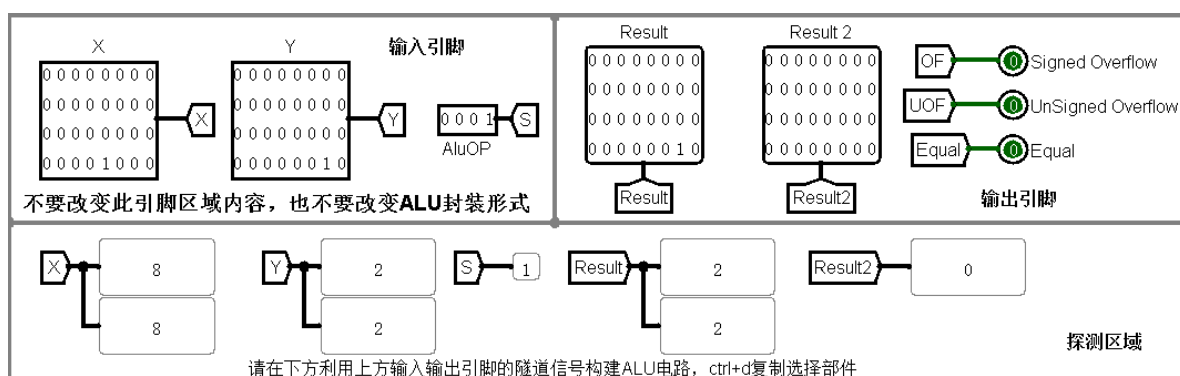


图 1-25 算术右移示意图

#15: 按位与图 1-26:

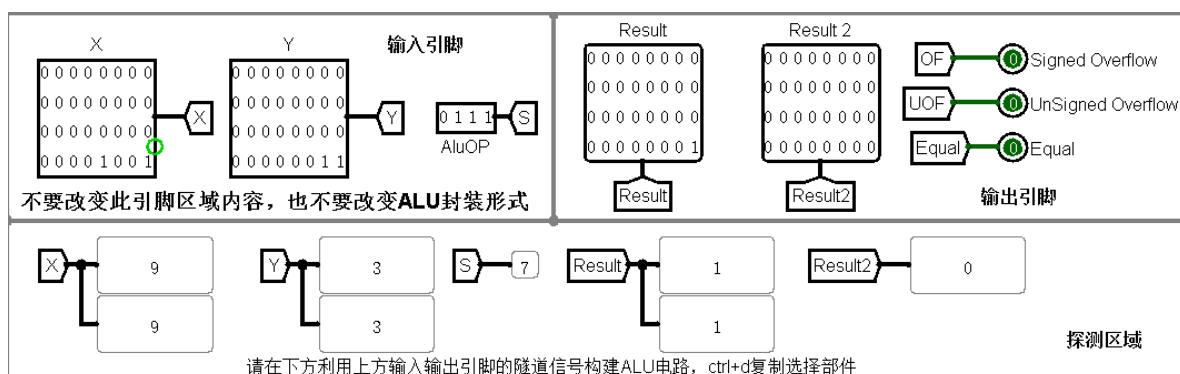


图 1-26 按位与示意图

# 华中科技大学课程实验报告

#16: 按位或图 1-27:

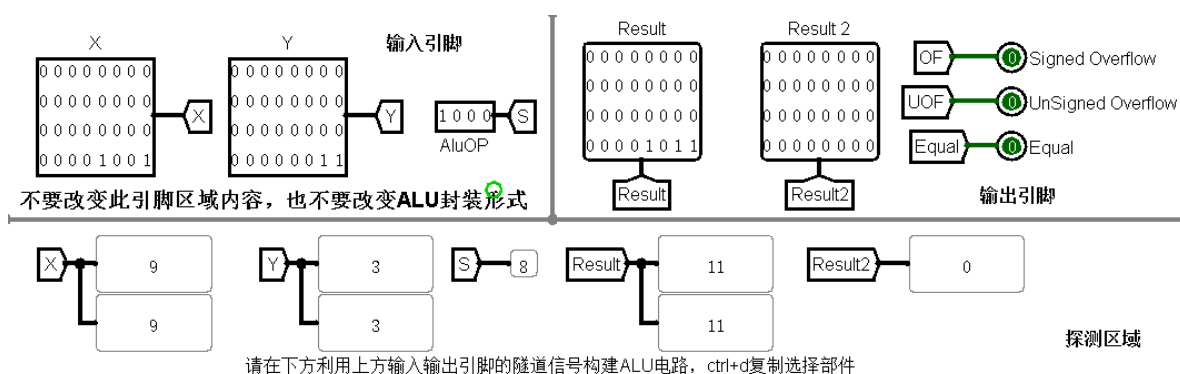


图 1-27 按位或示意图

#17: 按位异或图 1-28:

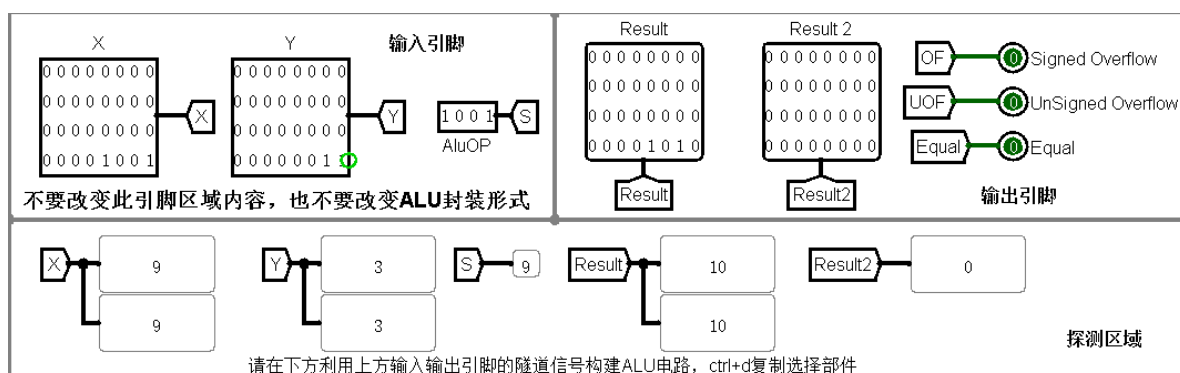


图 1-28 按位异或示意图

#18: 按位或非图 1-29:

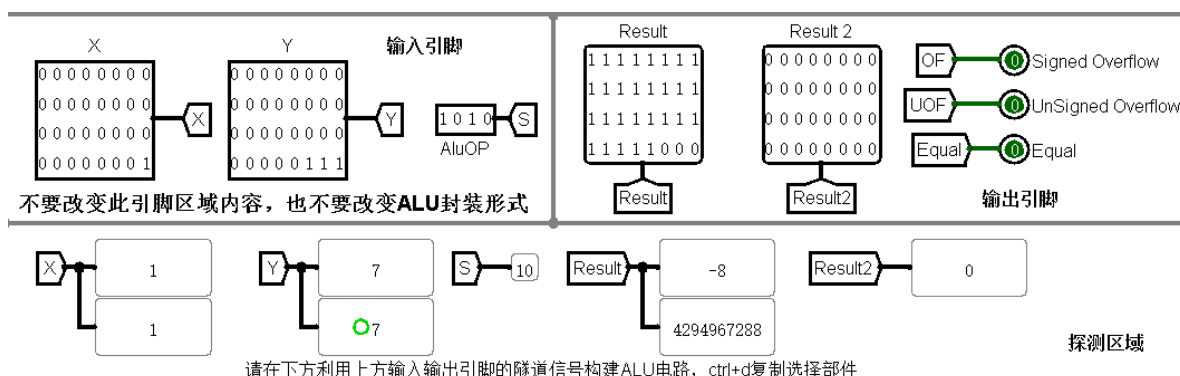


图 1-29 按位或非示意图

# 华中科技大学课程实验报告

#19: 无符号比较图 1-30:

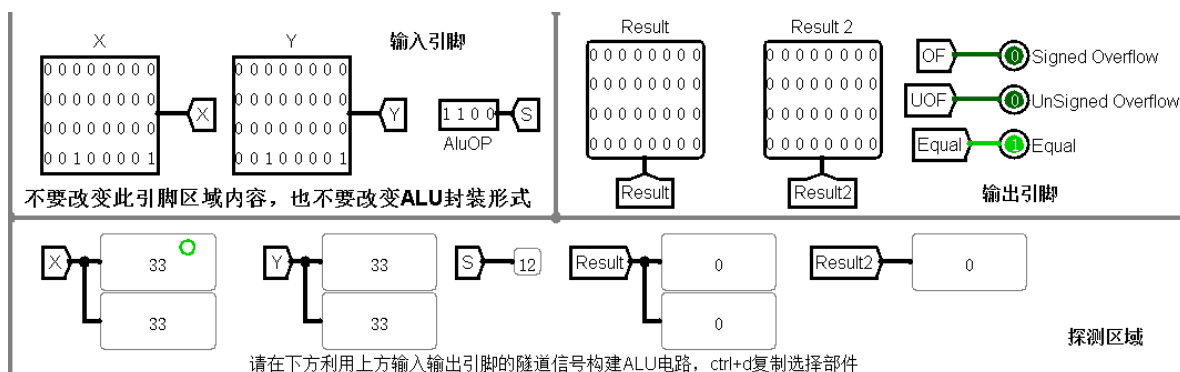


图 1-30 无符号比较示意图

#20: 有符号比较图 1-31:

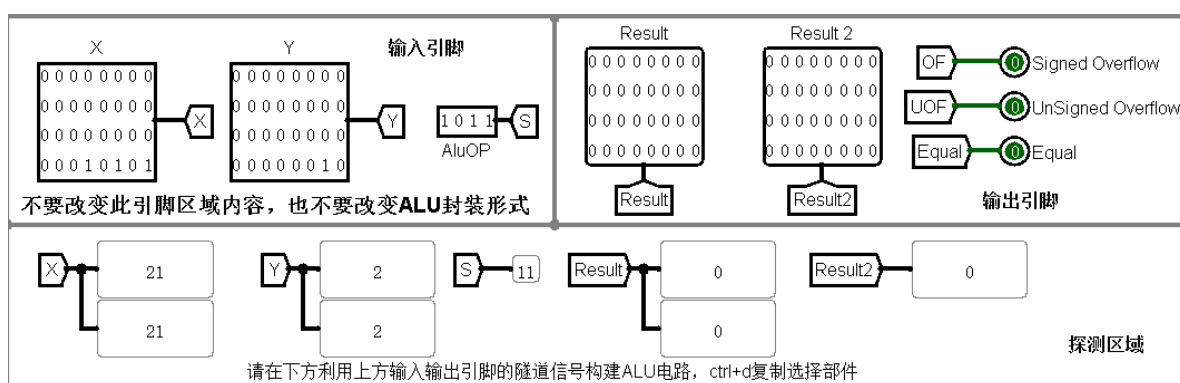


图 1-31 有符号比较示意图

所有功能测试均通过，ALU 搭建完毕。

## 2 存储器实验

### 2.1 设计要求

- 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，其具体功能如下表格 2-1，具体封装文件为 regfile.circ.

表格 2-1 芯片引脚与功能描述

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

注意零号寄存器值应该恒零！

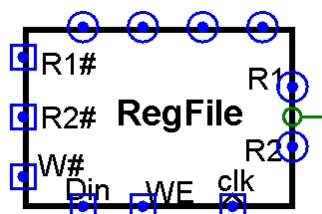


图 2-1 芯片引脚

- 2) 利用实验一封装好的运算器，以及 RAM 模块，封装好的 MIPS 寄存器文件，计数器等 logisim 模块构建一个自动运算电路，该电路由时钟驱动，可自动完成 RAM 模块（32\*32 位）0-31 号单元的累加，并将累加的中间结果回存到同一 RAM 模块 32-63 号单元。

## 2.2 方案设计

### 2.2.1 MIPS 寄存器组的构建

所有方案应将设计思路和设计原理、过程写清楚，为什么这样设计，各部件之间的关系，仅仅粘贴一张电路图是不合格的报告。

**总体思路：**MIPS 寄存器组的搭建是可利用 logisim 中已有的寄存器进行构建的，已有的寄存器搭建 32 个，然后将 32 个寄存器按照 8\*4 的排放规则排放，输入数据单口，选择单口，然后通过解复用器进行选择，选择完毕后将此寄存器写入使能端打开，将数据送入即可。

**输入处理：**输入寄存器组的信号肯定是 32 位的一个数据，输入数据的目的是将数据放到指定的寄存器，此时就要设计到指定寄存器，指定寄存器是通过 WE 端口输入的信号进行指定，此时就需要用到一个新的工具：解复用器。

解复用器的功能是将输入的数字解码成单个输出，例如一个 2 位的输入，可以解成四个输出端口，分别进行选择输出，一个使能端通过 WE 进行控制，实现寄存器的片选信号。

**输出处理：**输出是通过多路选择器实现的，因为共有 32 个输出端口，所以选择多路选择器，将多路选择器的输入接到各个寄存器的输出，选择信号是通过输入的信号进行确定的。

**连接处理：**寄存器组中共有 32 个寄存器，将 32 个寄存器按照 8\*4 进行排放，上下左右都空足够的距离，从左到右进行布线。

### 2.2.2 自动累加器数据同理实现

**整体分析：**自动累加器是通过实验 1 中构建好的 ALU 和实验 2-1 中构建好的 ram 进行中间结果寄存，将内存中的前 0-31 号单元进行累加，将累加结果送到 32-63 号单元中。



# 华中科技大学课程实验报告

---

**设计思路：**前 31 个单元进行累加，则可以采用钟控信号，时钟来时候，使用计数器进行累加操作，一直累加到 31，即可完成对 31 个单元的访问。一个节拍进行两个操作：前半节拍取累加数到寄存器组，后半节拍将计算好的结果送回到 ram 中，所以这儿的一个节拍需要的就是两个时钟周期。

经过分析，有如下的取值和累加和送回的关系对应：

取值的单元+31=送回累加和的单元

所以如果进行内存访问的时候，可以通过一个多路选择器进行实现，如果是前半节拍，则将计数值直接送入到内存进行内存读取；如果是后半节拍，则将计数器的值加 31 送到内存中，同时将内存写入打开，此时就可以顺利写会累加和了。

前半节拍取值，将值放到寄存器，后半节拍计算值，将累加结果放回内存中，这样就对内存进行了异步的访问，避免了可能存在的冲突的发生。

在计算完毕后，在利用一个多路选择器，将 16 位数码管的输出直接置为 63 号寄存器，显示最终结果，从而完成实验。

## 2.3 实验步骤

### （1）构造寄存器组

寄存器组首先需要进行布局的确定，因为寄存器组的部件，连线较为复杂，所以需要提前规划。

在进行仔细的推敲过后，确定了输入到输出从左到右，寄存器组 8\*4 的放置方法，即左边配置解复用器，中间放置寄存器，右边方式多路选择器，然后进行连线。在同一时间内每个寄存器的输入端实际都有数据，但是通过解复用器对使能端进行了控制，能实际写入到寄存器的只有需求写入的部分。

每个寄存器的指令端这次实验都没有用到，所以全部置 0。

寄存器组的设计较为简单，按照上述思路构建出对应的电路即可正常使用寄存器组。

### （2）构造数据通路

数据通路构建根据上边的思路，首先要确定一个“节拍”，即把两个周期进行计数，两个周期为一个节拍，第一个周期进行取数，第二个周期进行计算和送回，在

设计中的节拍计数是通过对脉冲沿进行计数，然后对计数的取反来实现的，在高电平和低电平分别执行两部分指令，构建原理如下图 2-2：

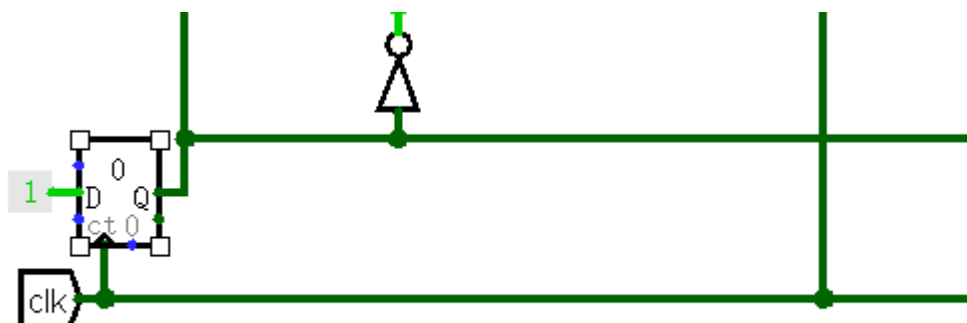


图 2-2 数据通路降频示意图

在每个节拍中，前半拍需要对寄存器的第 31 位进行访问，而后半拍对高位进行访问，高位访问和地位访问的区别仅仅在于访问地址的第六位（5）处的值不同，即高地址等于低地址加 32，所以可以通上边的计数触发器，因为在电平高的时候执行的是高地址而电平是低的时候执行的是低地址，所以将上边的脉冲计数器的输出端直接连接到访问地址的第六位（5）上边，而低位是通过一个模 32 的计数器来实现控制的，每个时钟计数一次，当检测到达到最高位的时候停止。

停止的方法是在 ram 的地址输入端口加一个二路选择器，如果检测到达了计算尾端，则发出停机指令，然后将地址的访问端口强制置为 63，电路即可停止了。

寄存器组的使用方法：在每次从内存 ram 单元中取出数字后，放入到寄存器组的指定两个单元中，在下半个节拍时候将这两个数据送到 alu 中进行计算，计算的结果送回 ram 中指定位置。所以寄存器组在这儿起到一个过渡，缓冲，存放中间值的作用。

将每次取出值和中间结果累加值一次次的送到寄存器中，然后在通过累加器进行累加操作，最后得出最终结果即可。

在最终结束的时候，检测到结束信号（计数达到进位），则触发输入到 RAM 的多路选择器，将输出选择为 RAM 的最高数据项，则保证了最后机器的停机和最终结果的正确显示

根据上述思路，即可较为顺利的完成本次实验。

## 2.4 故障与调试

### 2.4.1 解复用器 DMX 位数不匹配问题

**故障现象：**如图 2-3 将输入寄存器的值当做解复用器的输出，出现了位数不够用的问题



图 2-3 DMX 位数不匹配

**原因分析：**将数据当做解复用器的输出是一种错误的思路，实际上进行选择的是每个寄存器的使能端，使用数据是没法进行选择的。

**解决方案：**如图 2-4 将电路进行修改，将 dmx 的输入置为使能信号和输入寄存器编号信号，输出连接到各个寄存器的使能端。

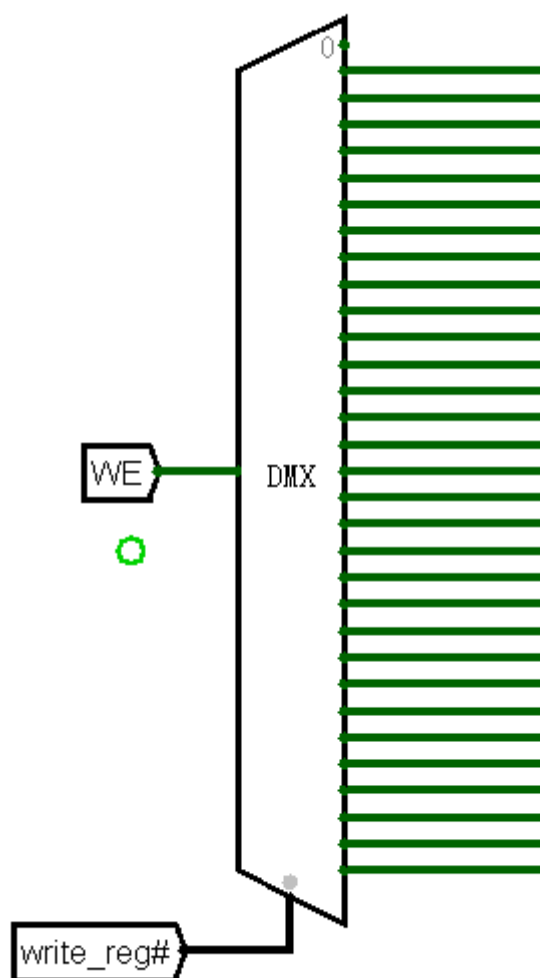


图 2-4 DMX 修改后效果图

## 2.4.2 30 号寄存器端不受使能端控制

**故障现象：**如图 2-5 在测试的时候，30 号寄存器无法接受到使能端控制。

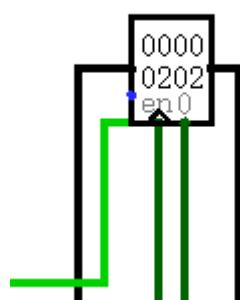


图 2-5 不受控制信号端

**原因分析：**在连线中，此处的使能端和输入的线没有连接到，导致信号无法输入。

**解决方案：**重新连线。

## 2.4.3 数据通路运算完毕后，显示每次的运算结果

**故障现象：**图 2-6 每次运算完毕，都会显示运算结果，而实验要求显示最终结果

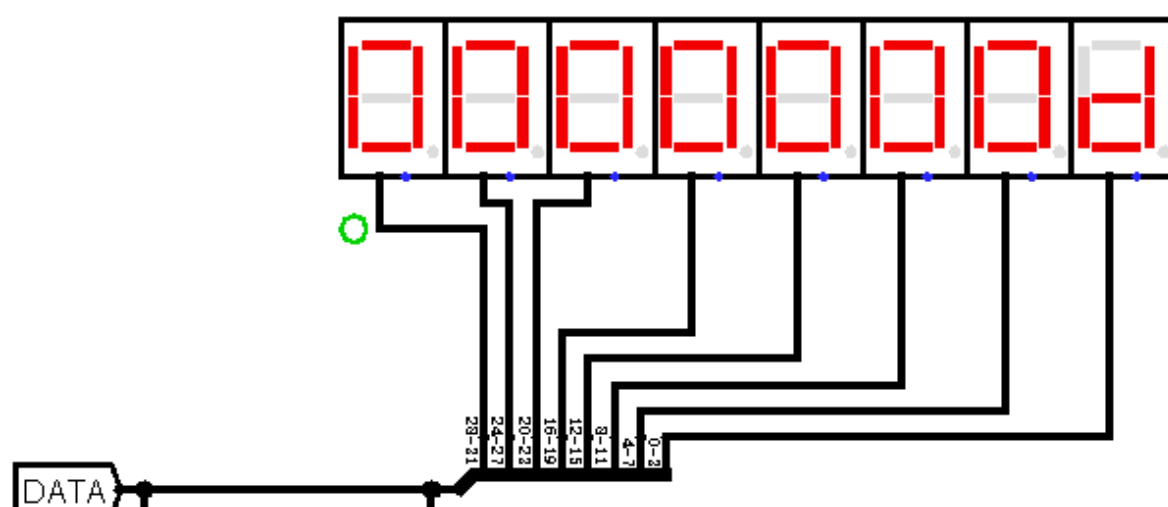


图 2-6 无法显示最终结果图

**原因分析：**十六位数码管显示是即时显示的，不受时钟的控制，所以每次的结果都会显示，所以应该在输出端口加一个判断条件

**解决方案：**如图 2-7 在输出端加一个三态控制。

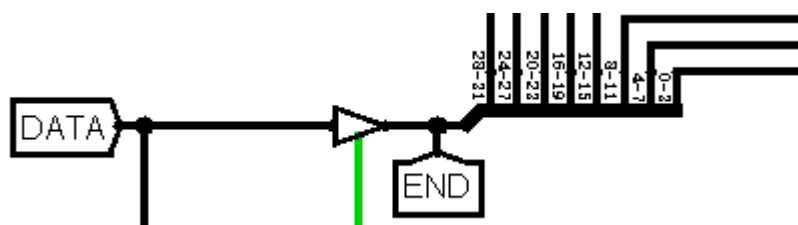


图 2-7 三态控制示意图

## 2.5 测试与分析

寄存器组的测试：

(1) 如图 2-8 当有输入加时钟信号但是无使能输入时：

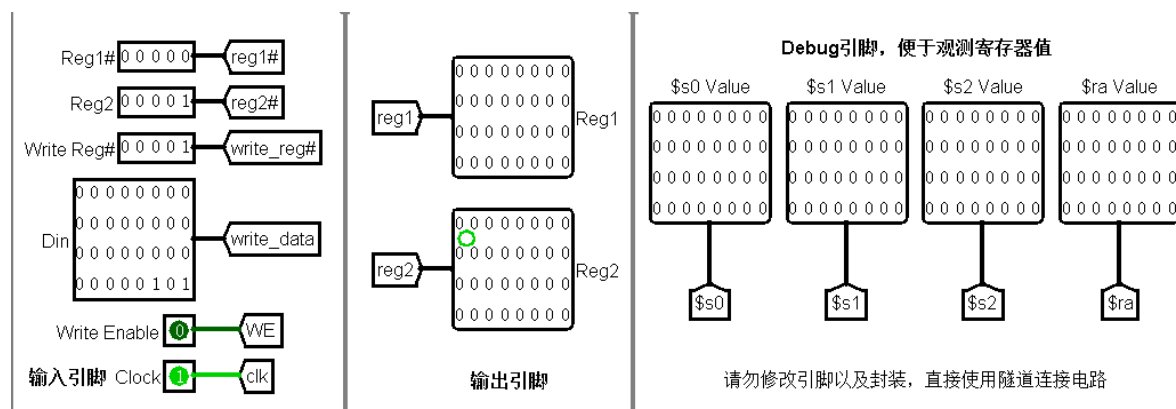


图 2-8 有时钟有输入无使能信号

(2) 如图 2-9 当有输入加使能无时钟信号时候：

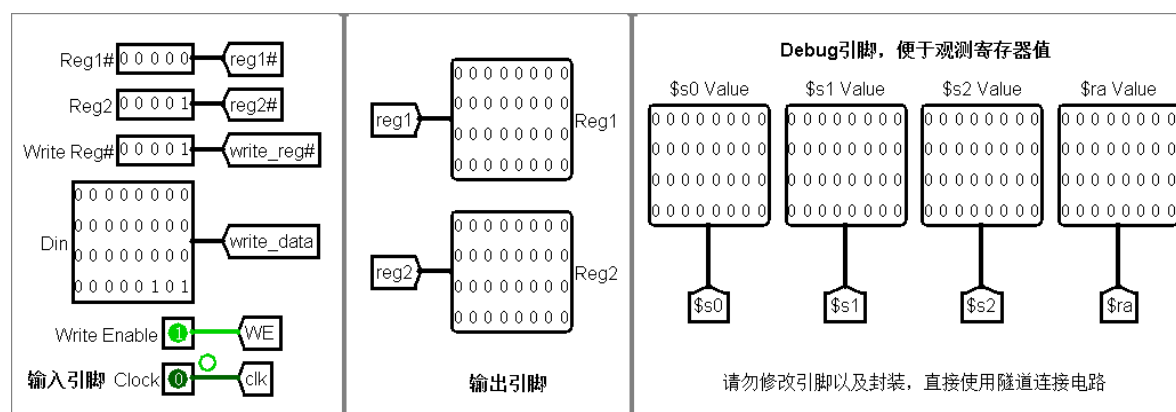


图 2-9 有输入有使能无时钟信号

(3) 如图 2-10 当有输入有选择有使能有时钟时候：

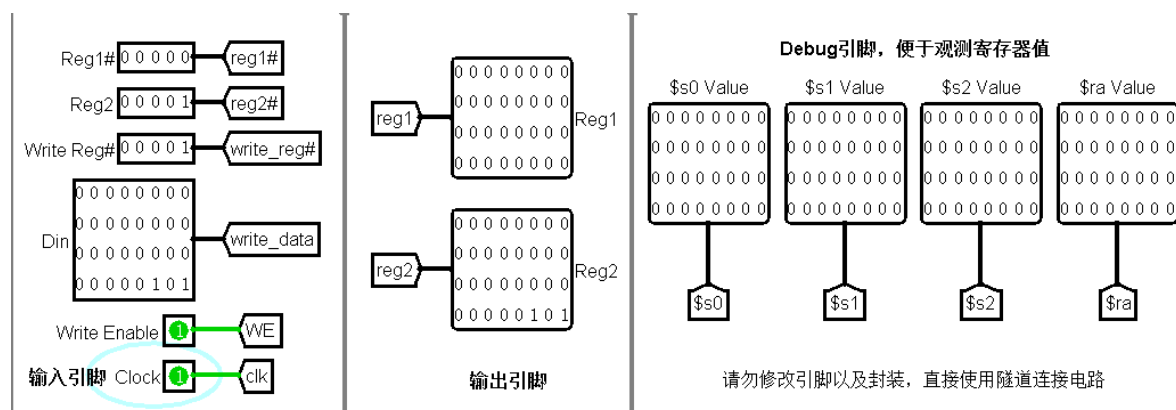


图 2-10 全部信号齐备时

(4) 如图 2-11ra 寄存器单独测试举例：

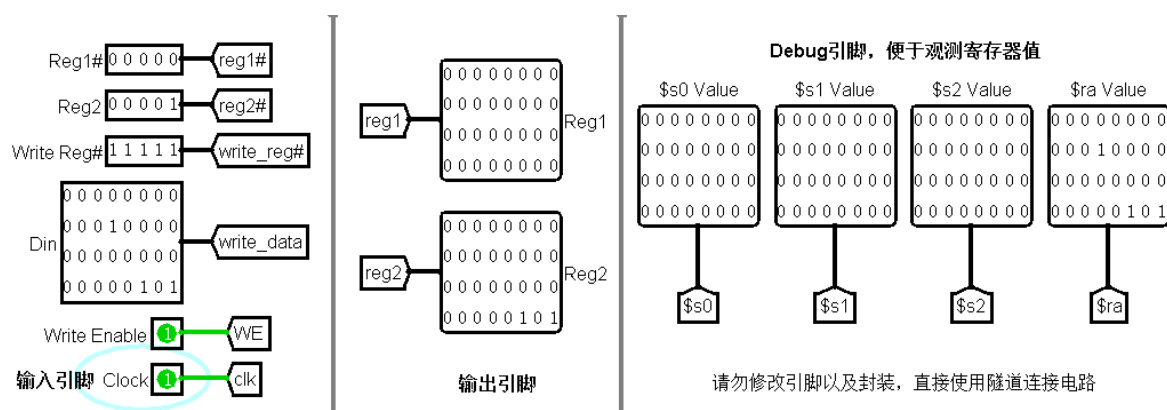


图 2-11 ra 寄存器单独测试图

自动累加电路测试：

加载 0-31 的镜像到 ram 的 0-31 位，然后调整合适的时钟频率，开始执行电路。

如图 2-12 执行过程中的 ram 查看：

00	00000000	00000001	00000002	00000003	00000004	00000005	00000006	00000007	00000008	00000009	0000000a	0000000b	0000000c	0000000d	0000000e	0000000f
10	00000010	00000011	00000012	00000013	00000014	00000015	00000016	00000017	00000018	00000019	0000001a	0000001b	0000001c	0000001d	0000001e	0000001f
20	00000020	00000021	00000022	00000023	00000024	00000025	00000026	00000027	00000028	00000029	0000002a	0000002b	0000002c	0000002d	0000002e	0000002f
30	00000030	00000031	00000032	00000033	00000034	00000035	00000036	00000037	00000038	00000039	0000003a	0000003b	0000003c	0000003d	0000003e	0000003f

图 2-12 ram 中间结果内容图

可以看到中间结果输出正常。

如图 2-13 在运算结束后查看 ram 中内容：



图 2-13 ram 运算内容查看图

可以看到，最后的累加结果正确，中间值也正确。，顺利通过测试。

# 华中科技大学课程实验报告

如图 2-14 查看 16 位数码显示管数值：

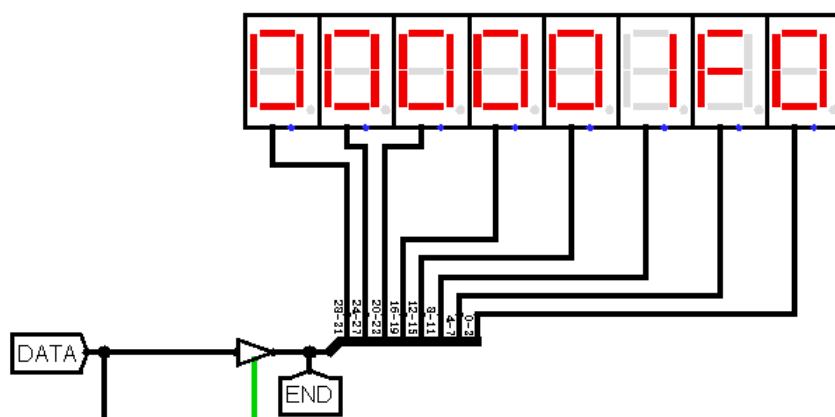


图 2-14 最终结果示意图

最终输出结果也是正确的。

寄存器顶层视图查看如图 2-15：

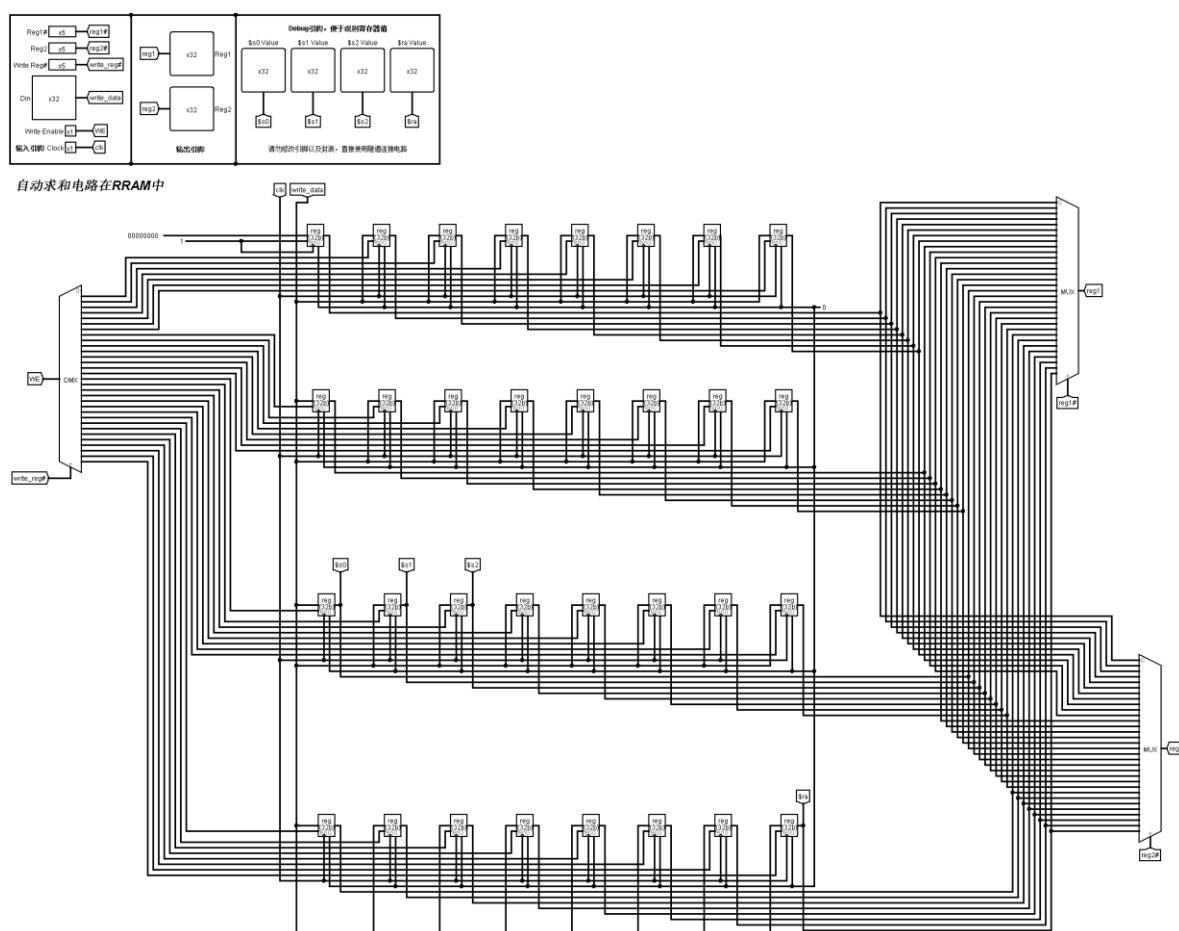


图 2-15 寄存器组顶层示意图



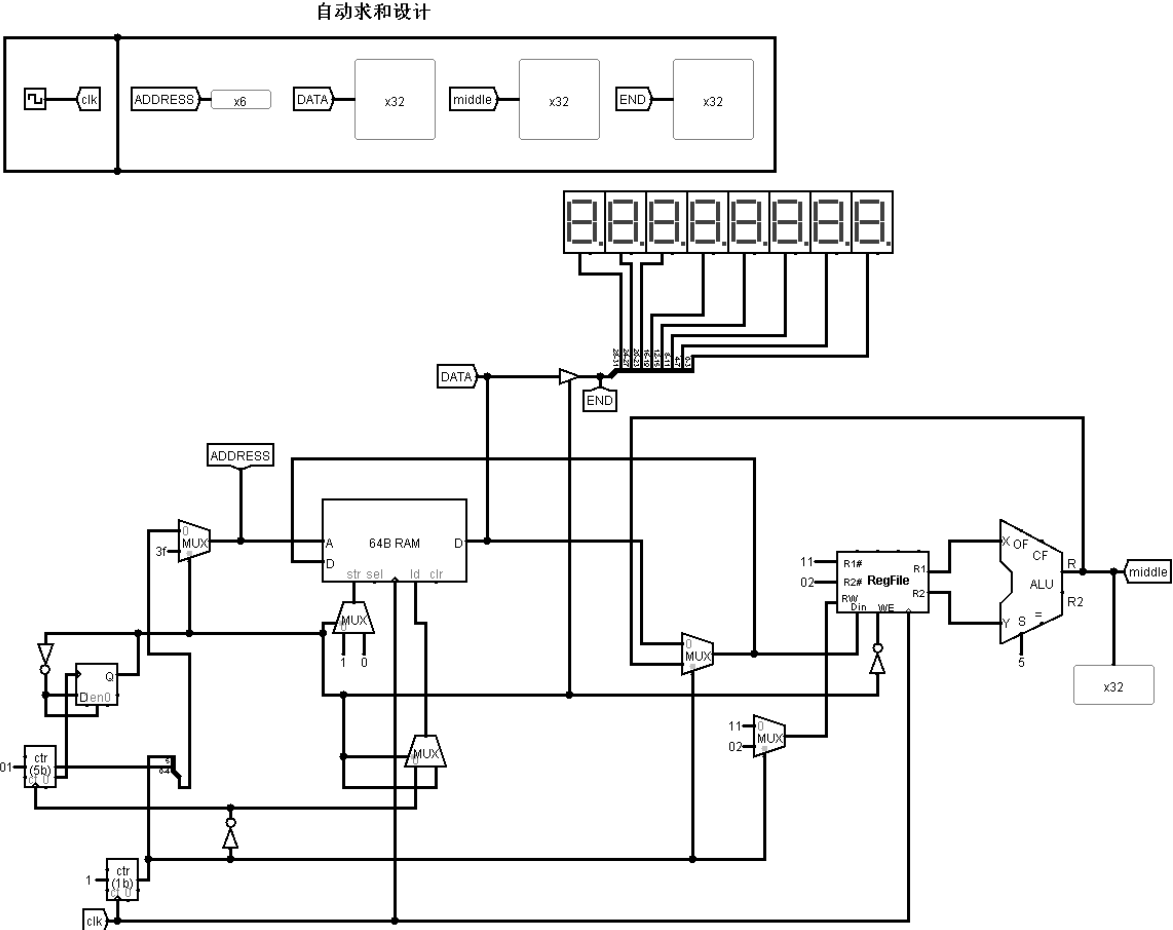


图 2-16 数据通路顶层示意图

实验二至此顺利完成。

## 3 CPU 实验

### 3.1 设计要求

利用实验 1、2 中构建的运算器，寄存器文件等部件以及 logsim 中其他功能部件构建一个 32 位 MIPS CPU 处理器，该处理器应支持下表中所述指令，具体指令功能参见附件中的 MIPS 标准文档。最终设计完成的 CPU 能运行教师提供的标准测试程序，程序存储在 logisim ROM 模块中（指令存储器、数据存储器分开）

学习汇编器 mars 仿真器，该仿真器功能强大。注意为了能让 mars 中汇编的机器码能在 logisim 中实用，需要设置 mars 界面中 setting 的 Memory Configuration，将内存模式设置为下图的模式，这样数据段起始位置就是 0 开始的位置。

利用 logisim 平台中现有部件构建取指令通路（PC 寄存器 32 位），其中 PC 由时钟驱动，每个时钟自动完成取值以及  $PC=PC+4$  的功能，控制存储器在后续步骤中实现。此处指令存储器选用 10 位地址线，32 位数据线的 ROM 部件，MIPS32s 位地址为字节地址，ROM 地址线宽度有限，建议将 CPU 32 位地址高位部分和字节偏移部分直接屏蔽，数据存储器也参照类似方法处理。

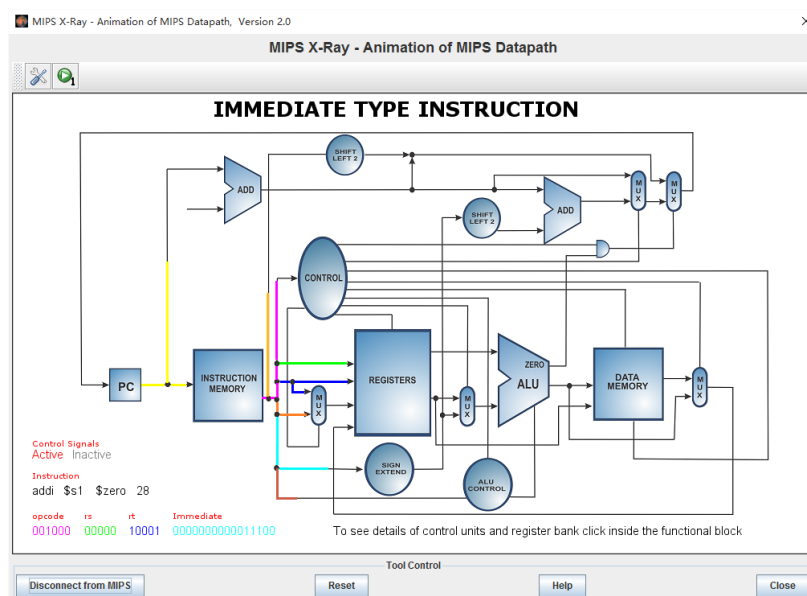


图 3-1 Mips x-ray 示意图

# 华中科技大学课程实验报告

表 3-1 指令格式

#	指令	格式	备注
1	Add	add \$rd, \$rs, \$rt	指令功能及指令格式 参考 MIPS32 指令集
2	Add Immediate	addi \$rt, \$rs, immediate	
3	Add Immediate Unsigned	addiu \$rt, \$rs, immediate	
4	Add Unsigned	addu \$rd, \$rs, \$rt	
5	And	and \$rd, \$rs, \$rt	
6	And Immediate	andi \$rt, \$rs, immediate	
7	Shift Left Logical	sll \$rd, \$rt, shamt	
8	Shift Right Arithmetic	sra \$rd, \$rt, shamt	
9	Shift Right Logical	srl \$rd, \$rt, shamt	
10	Sub	sub \$rd, \$rs, \$rt	
11	Or	or \$rd, \$rs, \$rt	
12	Or Immediate	ori \$rt, \$rs, immediate	
13	Nor	nor \$rd, \$rs, \$rt	
14	Load Word	lw \$rt, offset(\$rs)	
15	Store Word	sw \$rt, offset(\$rs)	
16	Branch on Equal	beq \$rs, \$rt, label	
17	Branch on Not Equal	bne \$rs, \$rt, label	
18	Set Less Than	slt \$rd, \$rs, \$rt	
19	Set Less Than Immediate	slti \$rt, \$rs, immediate	
20	Set Less Than Unsigned	sltu \$rd, \$rs, \$rt	
21	Jump	j label	
22	Jump and Link	jal label	
23	Jump Register	jr \$rs	
24	syscall (display or exit)	syscall	

If \$v0==10 停机  
else 数码管显示\$a0 值

# 华中科技大学课程实验报告

## 3.2 方案设计

### 3.2.1 指令译码

表格 3-2 指令功能表

OP 码	FUNCT 码	对应指令
000000	100000	ADD
001000		ADDI
001001		ADDIU
000000	100001	ADDU
000000	100100	AND
001100		ANDI
000000	000000	SLL
000000	000011	SRA
000000	000010	SRL
000000	100010	SUB
000000	100101	OR
001101		ORI
000000	100111	NOR
100011		LW
101011		SW
000100		BEQ
000101		BNE
000000	101010	SLT
001010		SLTI
000000	101011	SLTU
000010		JMP
000011		JAL
000000	001000	JR
000000	001100	SYSCALL

# 华中科技大学课程实验报告

指令译码的作用是将输入的指令码：（R 型指令包括 op 六位和 funct 六位、其他指令为 op6 位）翻译为对应的指令名字，24 条指令根据 mips 译码，都有相对应的指令码，具体如**表格 3-2**：

对照表，将相对的输入数据翻译成对应指令，即可完成指令译码。

注：在一段地址中，高 6 位代表的是 OP 码，底 6 位代表的是 FUNCT 码。

## 3.2.2 指令对应操作

24 条指令，每条指令都对应了一定量的操作（类似于微指令），例如一个 ADD 指令需要操作有：寄存器写入，ALU 加法信号产生等等，在对指令进行分析统计后，可以设计出如下操作信号控制表：

**表格 3-3 指令功能对照**

指令	PCNXT	MTOR	RD	MW	BR	ALUS	RW	ALUC	R1	SYS
ADD	00	00	01	0	0	00	1	0101	00	0
ADDI	00	00	00	0	0	01	1	0101	0	0
ADDIU	00	00	00	0	0	01	1	0101	0	0
ADDU	00	00	01	0	0	00	1	0101	0	0
AND	00	00	01	0	0	00	1	0111	0	0
ANDI	00	00	00	0	0	10	1	0111	0	0
SLL	00	00	01	0	0	11	1	0000	1	0
SRA	00	00	01	0	0	11	1	0001	1	0
SRL	00	00	01	0	0	11	1	0010	1	0
SUB	00	00	01	0	0	00	1	0110	0	0
OR	00	00	01	0	0	00	1	1000	0	0
ORI	00	00	00	0	0	10	1	1000	0	0
NOR	00	00	01	0	0	00	1	1010	0	0
LW	00	01	00	0	0	01	1	0101	0	0
SW	00	00	01	1	0	01	0	0101	0	0
BEQ	01	00	01	0	0	00	0		0	0
BNE	01	00	01	0	1	00	0		0	0
SLT	00	00	01	0	0	00	1	1011	0	0
SLTI	00	00	00	0	0	01	1	1011	0	0
SLTU	00	00	01	0	0	00	1	1100	0	0
JMP	10	00	01	0	0	00	0		0	0
JAL	10	10	10	0	0	00	1		0	0
JR	11	00	00	0	0	00	0		0	0
SYSCALL	0	0	0	0	0	0	0		0	1

## 3.2.3 控制单元的构成

控制单元 CLU (control unit) 是通过对上边叙述的 24 条指令进行“翻译”，得出这些指令需要干什么，然后将需求输出，外边链接各个控制单元的使能端，完成操作。

所以需要对上边列出的 9 个不同输出信号量进行解释：

PCNXT:指下一条指令的位置，PCNXT 共有 4 种状态，因为系统一共有四种地址转方式：PC+4（直接转移）对应状态 00，PC+位移（分支指令）对应状态 01， $PC=PC_{31..28}||instr\_index||0^2$ （jmp 和 jal 部分）对应状态 10，PC=[RS]（跳转寄存器）对应状态 11。

MTR:memory to regfile 的简称，即内存中的数据读取到寄存器中，读取到的位置（RS,RT,RD）由输入码确定。

RD:reg distance 即寄存器地址，因为指令分为三种（J 型 R 型 I 型），每种类型对应的访问寄存器位段不同，所以需要进行区别对待。

MW: memory write 内存写入判断，为 1 代表需要写入内存。

BR: branch 分支指令判断

ALUSRC:判断第二个操作数的来源，因为指令的不同类型而需要。

RW:reg write 代表是否需要写入寄存器（大部分指令需要）。

ALUC:判断需要运算器进行什么功能。

R1:判断第一个操作数的来源（RS,RT）。

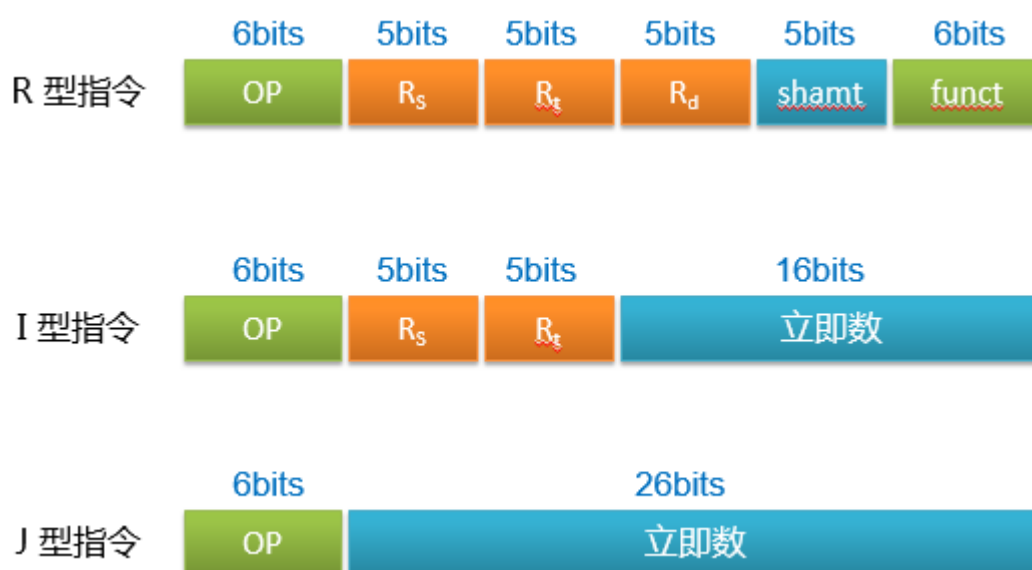
然后按照需求将表中内容实现就可以构成控制器单元。（本实验主要完成的部分）

SYS:是系统调用的判断出口，如果有系统调用，则 sys 端口为 1，否则为 0，在输出控制单元的外部判断是否有 sys 信号产生，如果的话再进行相应的寄存器判断，最终实现最后的功能（显示、停机）等。

对指令的设计实际上是参照了微指令的设计思路加硬布线的设计方法，整体布局错落有致，保证了延迟不超过 3 个门级时间而且也保证了电路的简洁美观。

控制单元的设计没有采用分析电路，而是手工连线。

## 3.2.4 JRI 型指令格式的区别



## 3.2.5 寄存器单元的构成

本实验的寄存器单元直接调用实验二完成的寄存器组即可。

## 3.2.6 运算器单元的构成

本实验的运算器采用实验一完成的运算器。

## 3.2.7 地址跳转设计

地址跳转利用在控制单元中已经输出的 PCNXT 输出，将 PCNXT 当做多路选择器的选择端，输入端接各种对应跳转方式，输出接 PC 累加端就可。

对应的设计图 3-2:

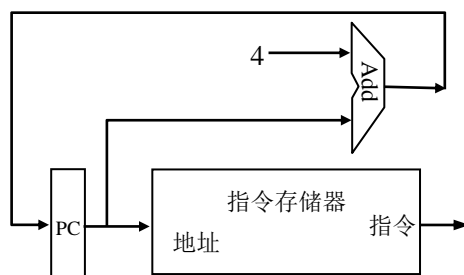


图 3-2 跳转指令设计图

## 3.2.8 信号发生器设计

信号发生器是通过对输入时钟信号 `clk` 做逻辑操作实现的，因为需要判断是否停机，当停机时候是没有时钟信号发生的，所以对时钟信号应该进行二路选择，如果停机指令 `halt` 发出，则停止产生时钟，设计原理如下图 3-3:

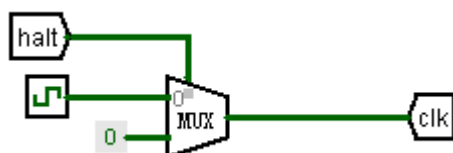


图 3-3 信号发生器设计图

## 3.2.9 Syscall 信号处理

Syscall 信号是系统功能调用信号，如果此信号来到，则需要判断：

If `$v0==10`

`halt`(停机指令)

else 数码管显示 `$a0` 值

所以可以得出 `syscall` 指令图 3-4:

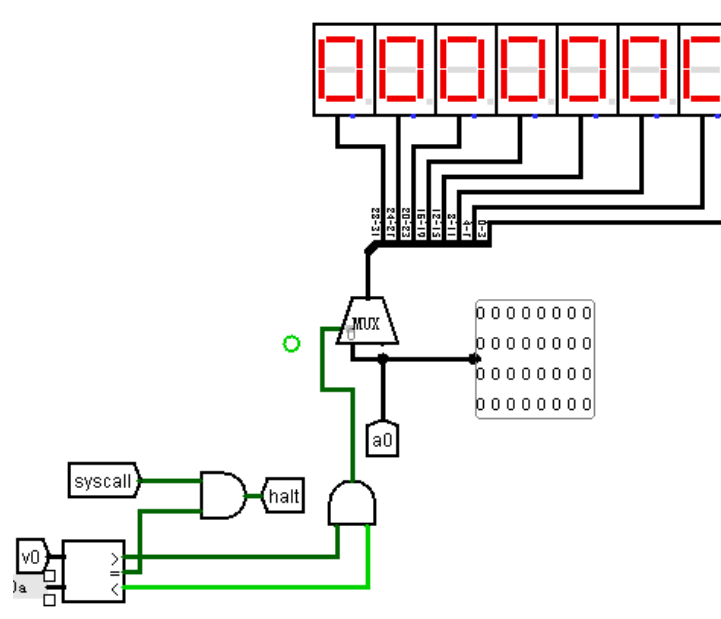


图 3-4 `syscall` 指令调整图

`Halt` 信号即为停机信号。



## 3.2.10 操作数的选择

操作数的选择是通过多路选择器来实现的，不同地点的不同操作数，在 **clu**（控制单元）中已经进行了划分，所以只需要将划分标准连接到相应的多路选择器选择端口即可。

## 3.2.11 R 型指令的数据通路

R 型指令是寄存器—寄存器型的指令，该类型的指令整体数据通路相同，设计方法如下图 3-5：

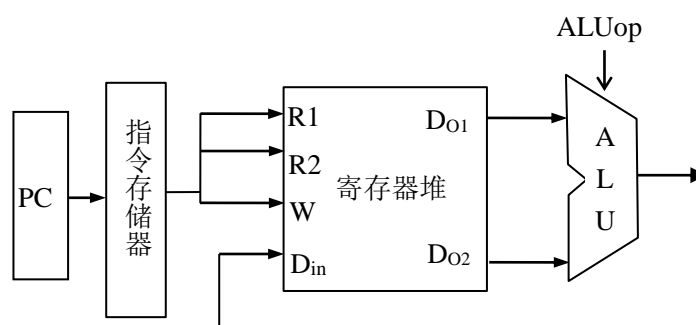


图 3-5 R 型指令的数据通路图

## 3.2.12 I 型指令的数据通路

I 型指令是立即数型指令，该类型的指令整体数据通路相同，设计方法如下图 3-6：

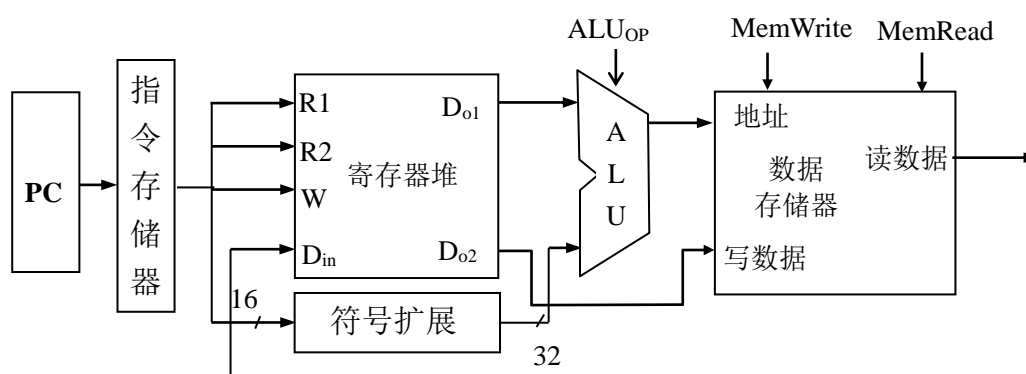


图 3-6 I 型指令数据通路图

### 3.2.13 J型指令的数据通路

J 型指令是跳转型的指令, 该类型的指令整体的数据通路相同, 设计方法如下

图 3-7:

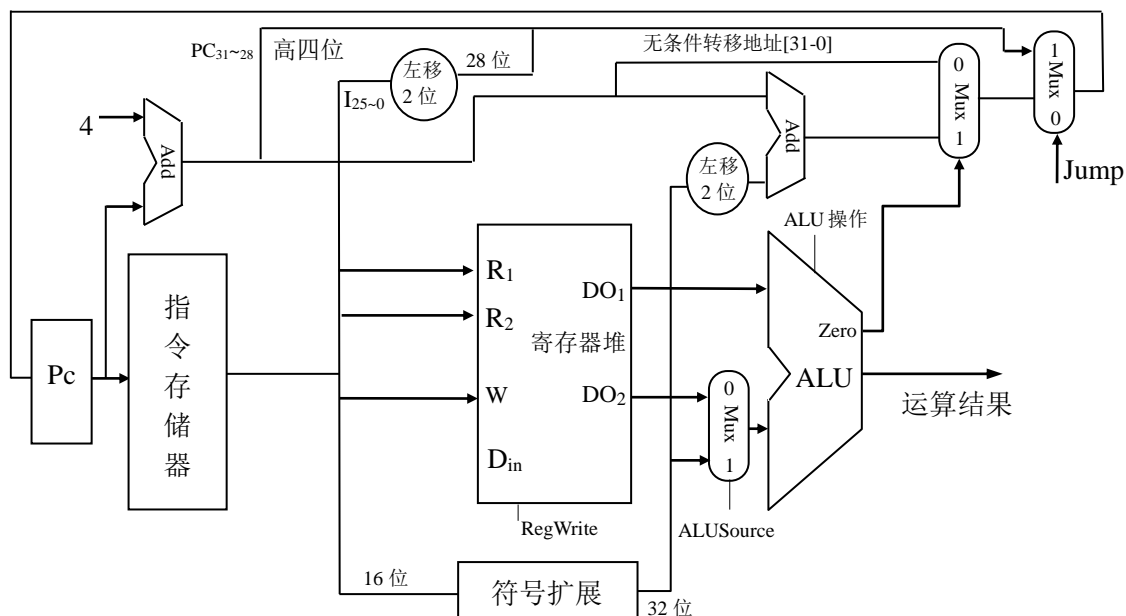


图 3-7 J 型指令数据通路图

### 3.2.14 指令的计数方式

指令计数分为总指令条数统计和各种类型指令条数统计。

因为本次实验完成的是单周期指令的 CPU，即一个时钟周期对应一条指令，所以统计指令条数就是统计在系统发出 `halt` 指令前的时钟周期数（包括 `halt` 在内）。

所以可以得出总指令条数的设计如下图 3-8:

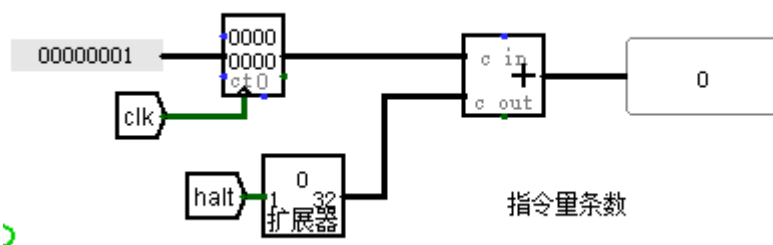


图 3-8 指令量条数计数图

该系统完成的功能是在统计 halt 指令前的指令条数后，在单独加 halt 指令，得出最终指令条数。

# 华中科技大学课程实验报告

单独指令的计数控制端由 CLU（控制单元）译出指令后根据指令的种类得出：  
其中指令分类五类指令，分别是：

将这些信号作为使能端，对时钟进行计数，因为 OTHERS 中包含一个 syscall 指令，这个指令中存在 halt 指令可能会导致 others 指令计数少一条，所以需要在这个计数结束后，对 others 单独加 1。

单周期 CPU 的构成基本较为简单，设计高层视图如下图 3-9:

图 3-9 单周期 CPU 高层视图

## 3.3 实验步骤

### 3.3.1 指令译码器构建

指令译码器是根据 24 条指令的操作码将其翻译成 24 条指令，每条指令的确定是通过 12/6 位的传入的地址码来确定的，将输出表示成 12/6 位的真值表达式，输入分析电路中，利用 logisim 中的自动生成电路，产生表达式，使用方法如下图 3-10:

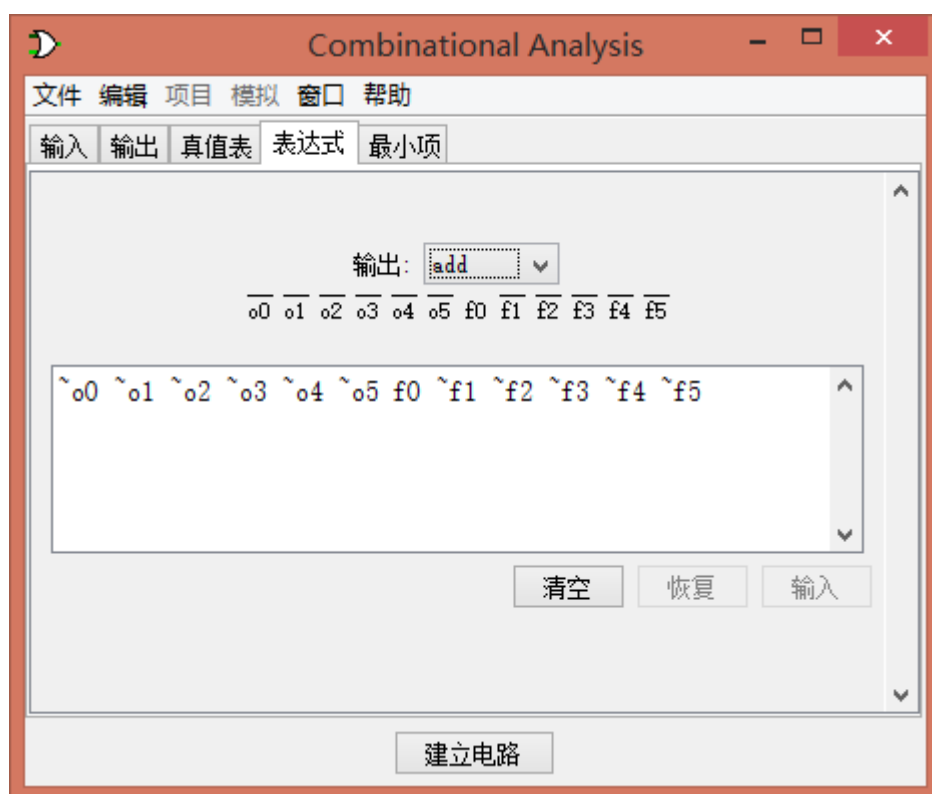


图 3-10 分析电路使用方法图

例如 add 指令对应的操作码是 000000 100000 (op funct)，所以输入的信息就是： $\sim o0 \sim o1 \sim o2 \sim o3 \sim o4 \sim o5 f0 \sim f1 \sim f2 \sim f3 \sim f4 \sim f5$ 。

由于 logisim 软件分析生成电路最大只支持 12 条而我们有 24 条指令，所以将 24 条指令分成两组，每组 12 条，在两个电路中生成分组，然后合并成一个电路，最后部分使用隧道，达到简化电路的目的。

最终生成的译码器外观如下图 3-11:

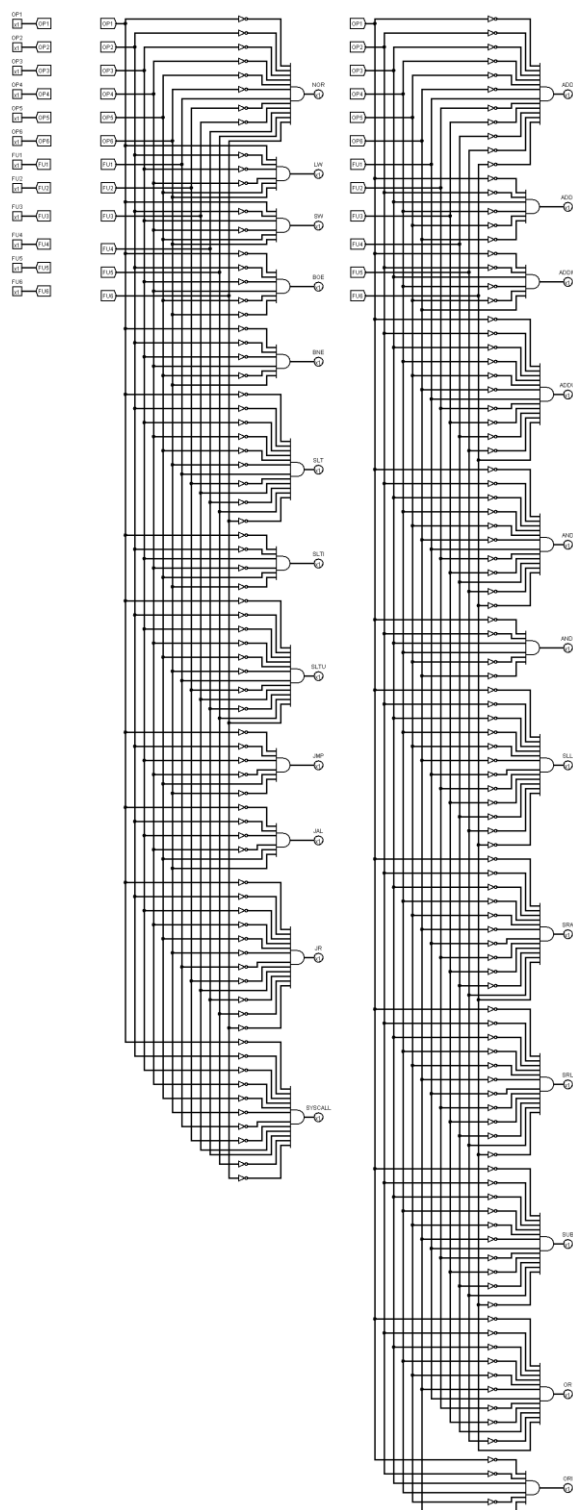


图 3-11 译码器外观图

## 3.3.2 控制器单元的搭建

控制器是建立在指令译码器的基础上的，在生成了译码器 decoder 后，将译码器封装芯片放到控制单元中，decoder 的输入端是输入的 12/6 位操作码，经过译码后，输出是 24 个指令，然后根据设计思路中的搭建表格，将各个类似“微指令”的输出全部连接，最后完成指令统计的输出。

每个指令都对应相对来说较为估计的微操作，将指令翻译成微操作，输出到各个端口，这便是控制单元的职责。

设计概图如下图 3-12:

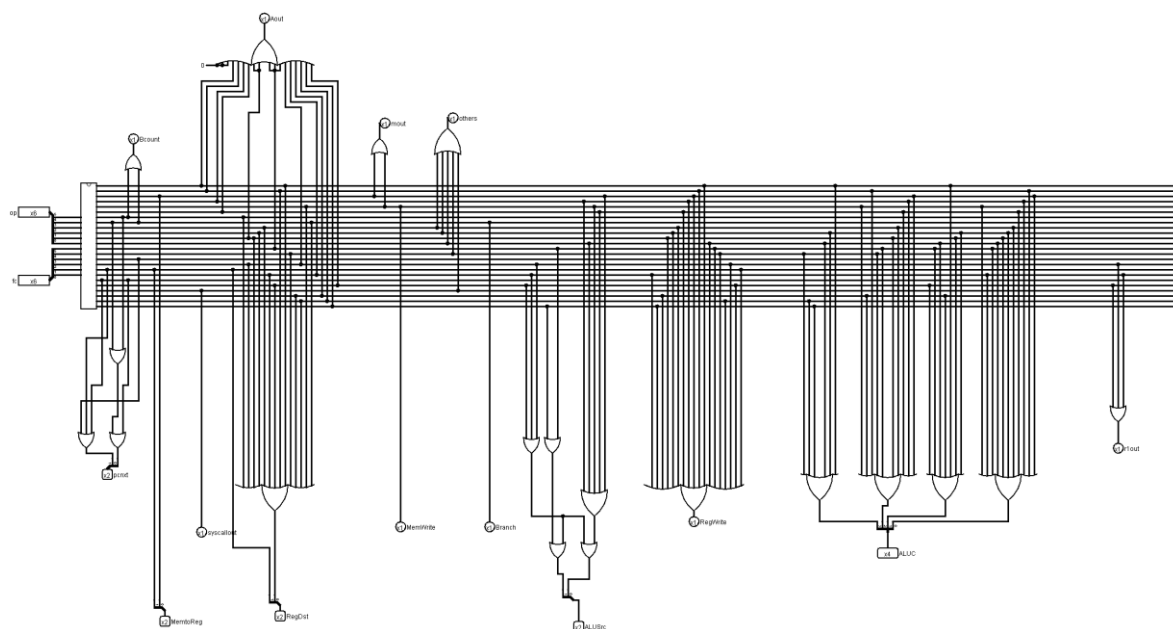


图 3-12 控制单元设计图

其中左端为输入，输入时 OP 和 FUNCT 码，然后将它们送到指令译码器中译出 24 条指令，然后将 24 条指令对照上边微指令表进行连线，如果需要某条微指令，则将其输入到该微指令的或端，下边为微指令输出，而上边为指令输出，微指令输出如果需要多位的则用分线器将多位一一分开，哪一位需要使用则把它连入到相应位（因为默认无输入时候输出值为 0），指令计数是通过控制使能端来实现的，如果该指令到来，则将相应的输出端控制为 1，将上层电路指令计数的累加器激活，完成计数。

## 3.3.3 beq bne 指令

beq bne 指令是通过控制器的 branch 指令拉出来的，使用运算器的判断相等终端，判断是否相等（或不等），如果相等（或不等），则启动相应的跳转。

设计图如下图 3-13:

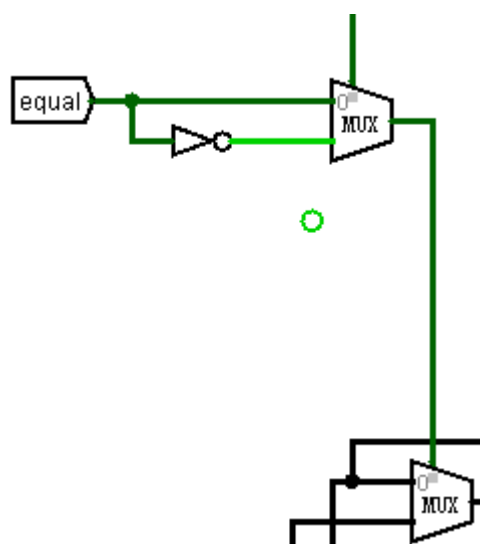


图 3-13 BRANCH 指令设计图

上端输入 beq/bne 产生指令，如果是 bne，则调用多路选择器的 1 端，判断是否满足 bne，满足则触发下边的多路选择器，实现跳转。如果是 beq，则触发上边多路选择器的 0 端，触发了 0 端后即可判断是否满足 beq 的条件，如果满足则触发下边的多路选择器，完成 beq/bne 的设计。

## 3.3.4 整体地址计数的设计

整体的地址计数分为四种情况，一是普通计数，二是分支计数，三是跳转计数，四是跳转到寄存器计数。分别对四种情况进行设计。

第一种是普通顺序指令执行，由于是 32 位 mips 机器，所以每条指令长度 4 个字节，每次 pc 执行后，pc 值加 4 再送回即可，但是由于这儿是单周期 CPU，必须再连接一个加法器来实现 pc 累加，不可以通过 alu 来实现。

第二种是对分支指令的计数，分支指令是 beq, bne，它们的设计在上边已经叙述完毕，这儿不再赘述。

第三种是跳转指令（JMP,JAL）它们两个实现的是将跳转的地址通过 PC+4 再进





顶层模块的参考设计图如下图 3-15:

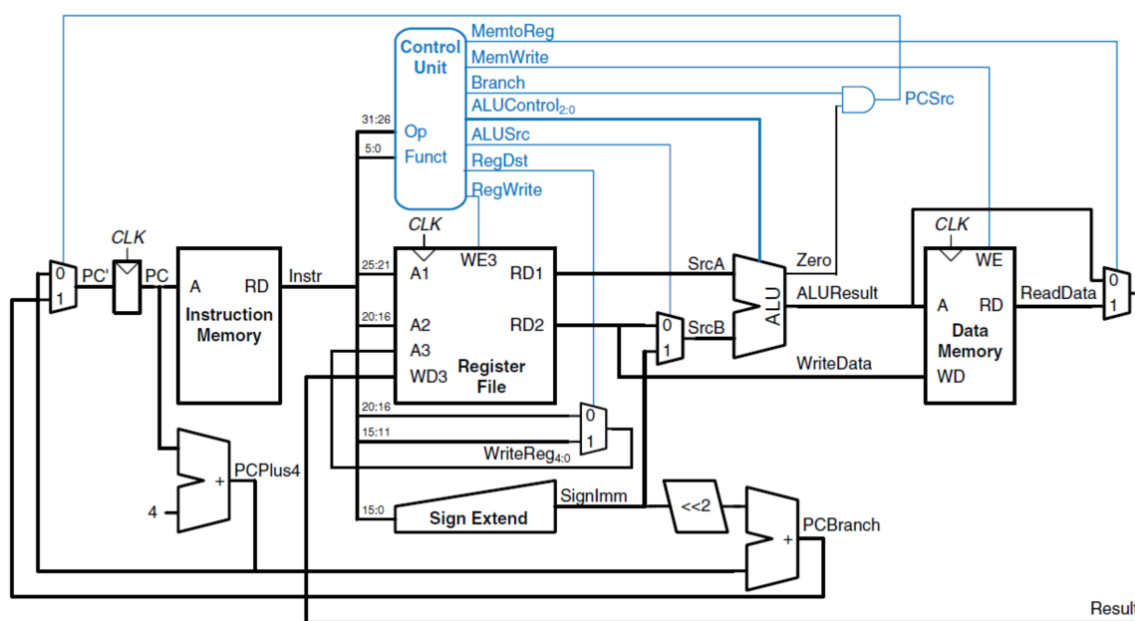


图 3-15 单周期 CPU 顶层模块示意图

在顶层模块的基础上，加入指令计数功能，指令计数功能是通过计数器实现的，计数器的使能端是各个指令或出来的，计时器的输入端接入时钟信号 `clk` 即可。

注：实际的实现比这个电路图要复杂一些，因为该示意图中指令个数远远不足 24 条，而我们设计的是支持 24 条指令的 CPU，所以需要加入很多数据选择器，来对指令译码生成的微指令进行控制，从而实现 CPU 的最终功能。

指令存储器使用 **ROM**，可以满足多次测试中间数据不需要重复调用，而数据存储器使用 **RAM**，可以满足多次测试中间数据不会互相影响，寄存器组调用实验 2 所完成的寄存器组，而运算单元也是调用实验 1 所完成的运算单元进行操作，但是因为是单周期 **CPU**，各种部件在一个周期内不可能被多次使用，所以需要考虑器件的个数。

## 3.4 故障与调试

### 3.4.1 NOR 指令译码出错

故障现象：如图 3-16，NOR 指令译码不是期望值。（输入为 000000/100111 下）

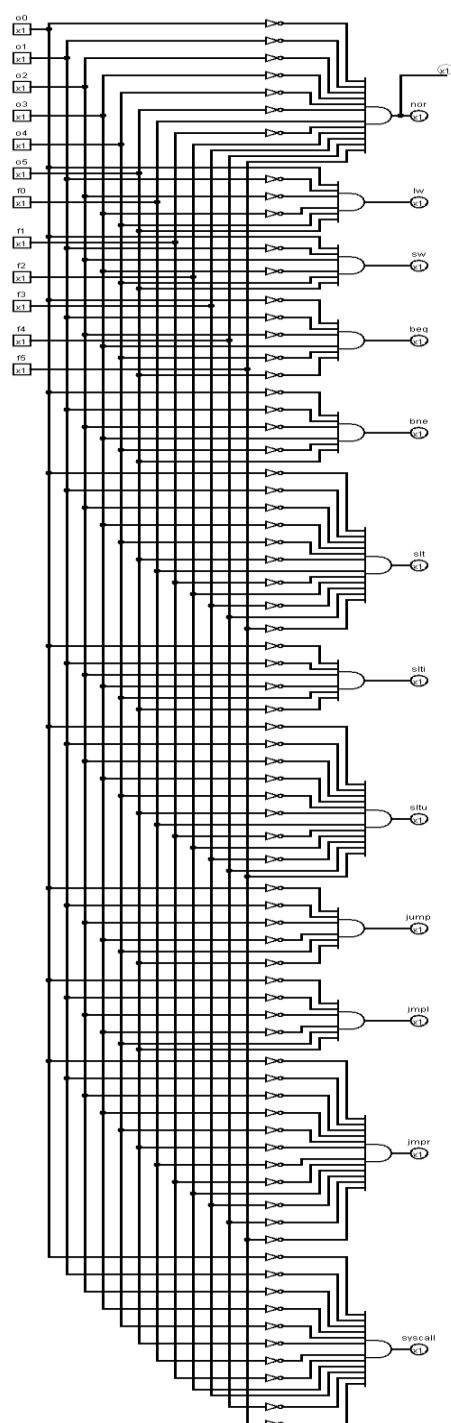


图 3-16 NOR 译码错误图

**原因分析：**如图，在输入分析电路时候，输入表达式错误。

**解决方案：**将输入表达式改正即可。

## 3.4.2 控制单元中输入相应操作码没反应

**故障现象：**如图 3-17 在构造好 CLU（控制器单元）后，输入相应的操作码无相应的指令信号输出。（原本输入的是 001000 是 ADDI 指令，但是却译码成了 BOE(BEQ)指令）

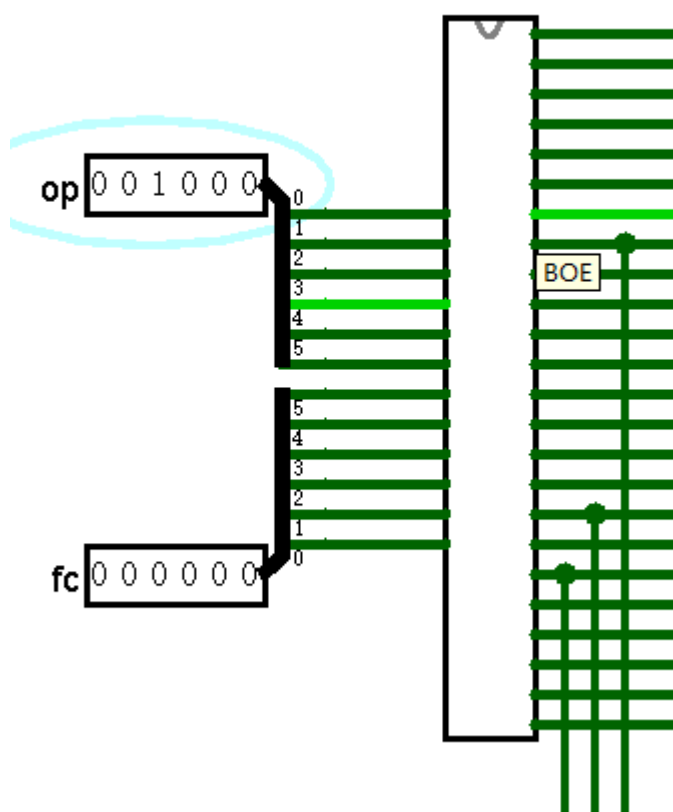


图 3-17 操作码错误图

**原因分析：**ADDI 的真实操作码是 001000 而 BOE 的真实操作码是 000100，两个刚好位数相反，即在将 OP 接入时候，与解码芯片 decoder 的连接放反了（0-5 顺序连接成了 5-0）。

**解决方案：**将 OP 的位数属性进行对调。

## 3.4.3 SLA,SRA,SRL 指令出错

**故障现象：**三个指令的输出只会有“一半”信息。（调试时候截图丢失）

**原因分析：**这三个指令的操作数地址在 rt 和 rd 而不是 rs 和 rd，在写时候写成了 rs 和 rt，导致位移指令出错。

**解决方案：**如图 3-18 在控制器中加入判断标记 r1out，它的作用是判断是否是这三条指令，如果是，则将输入 alu 的 x 设置为 1

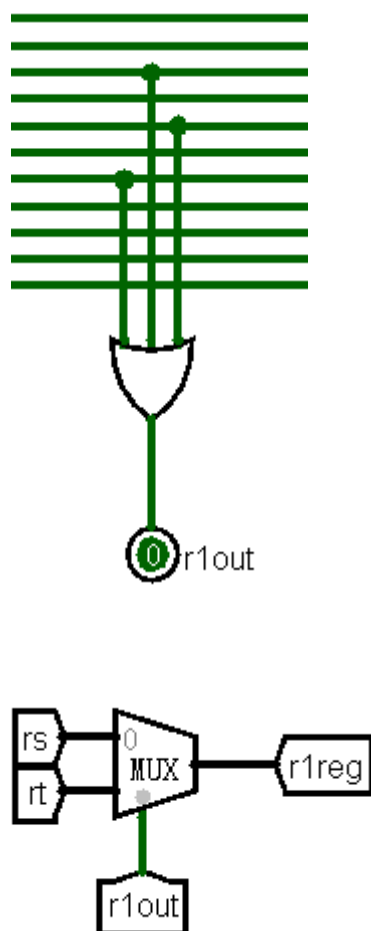


图 3-18 位移指令修复图

## 3.4.4 地址出错

**故障现象：**如图 3-19 CPU 无法正确的识别地址

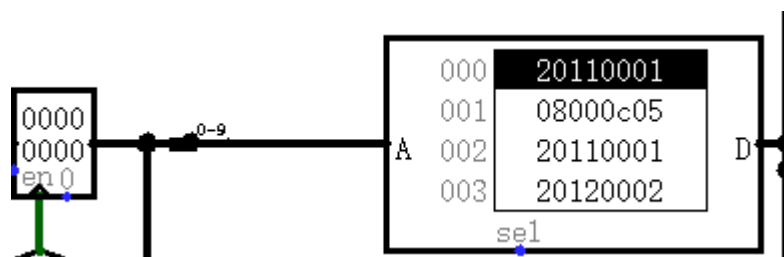


图 3-19 CPU 地址识别错误图

**原因分析：**在开始时候没有注意地址的格式，直接选用了低 10 位（0-9）位作为地址，从而导致地址识别错误

**解决方案：**如图 3-20 在查找出问题之后，直接用单位分线器将 2-11 位分出即可。

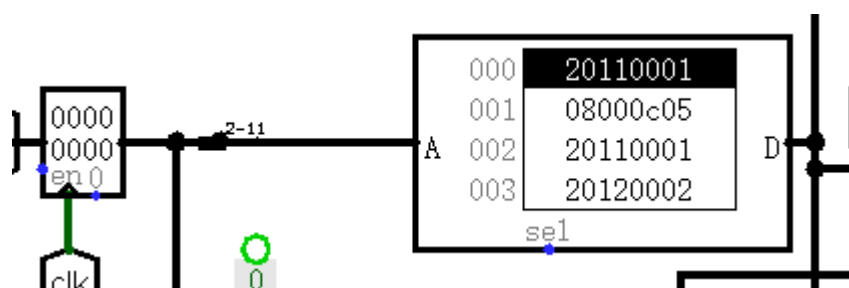


图 3-20 CPU 地址识别错误修复图

## 3.4.5 指令计数在最后总会少一条

**故障现象：**如图 3-21 在程序执行完毕后，应该有 1546 条指令，然而只会计数 1545 条。

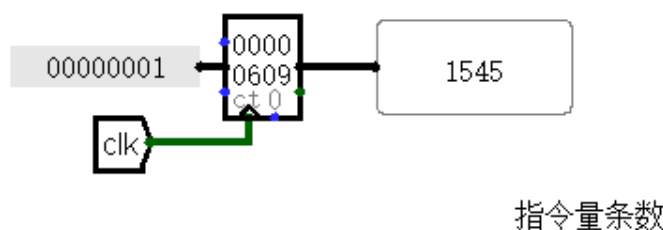


图 3-21 指令计数量错误图

**原因分析：**在指令条数计数的时候，没有将停机指令 halt 指令计入到其中。因为停机指令来到的时候，就将时钟 clk 停了，此时就不会再进行继续计数，从而导致了无法统计 halt 指令。

**解决方案：**如图 3-22 在查找出问题之后，在指令最后加 1（加 halt 指令）因为在一次运行中停机指令必定会在最终出现一次，所以在程序结束时候将指令加进去即可。

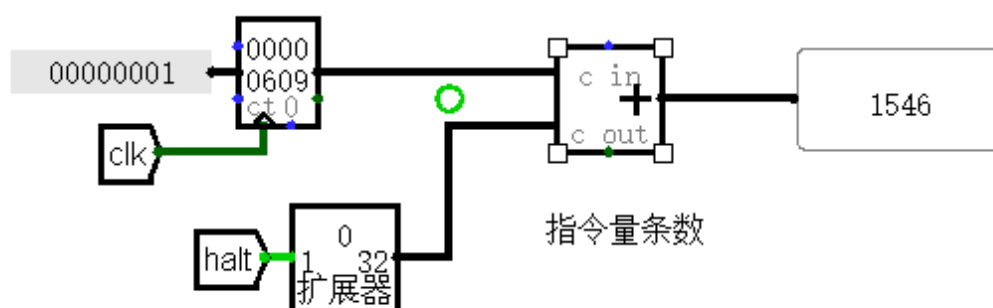


图 3-22 指令量计数错误修复图

## 3.4.6 其余错误

本次实验从开始编写到调试完成耗时足够 18 个小时，其中遇到了很多错误，但是大多都没有记录，如果将其列出来将会是一个工作量特别庞大的事情，所以只是调了部分有代表性，印象较为深刻的错误列了出来。

## 3.5 测试与分析

CPU 实验中对各条指令搭建的时候都进行了测试，但是没有进行详细的记录也没有代表性。所以这儿只使用了较为有代表性的 benchmark.asm 进行了测试：

首先将 benchmark.asm 调入到 mars 中进行反汇编：

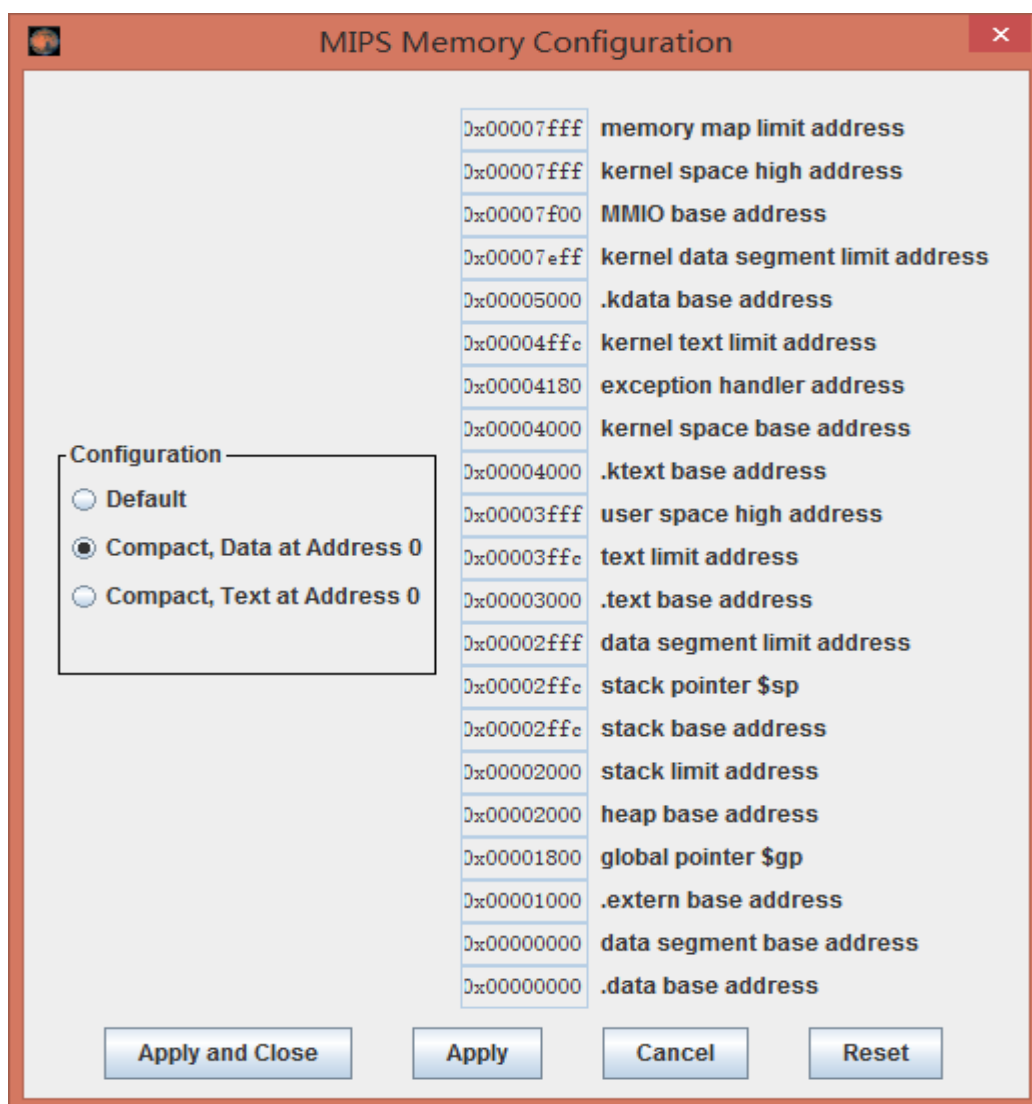


图 3-23 地址设置示意图

设置起始地址为 0，然后汇编：

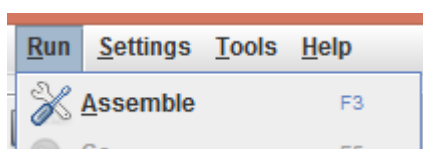


图 3-24 反汇编操作图

# 华中科技大学课程实验报告

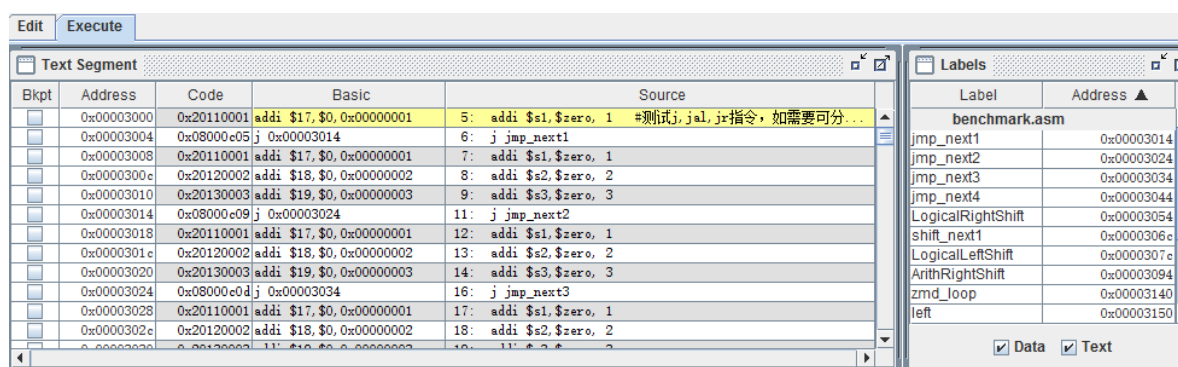


图 3-25 反汇编部分结果图

然后生成可供 logisim 的指令 rom 可以加载的镜像：

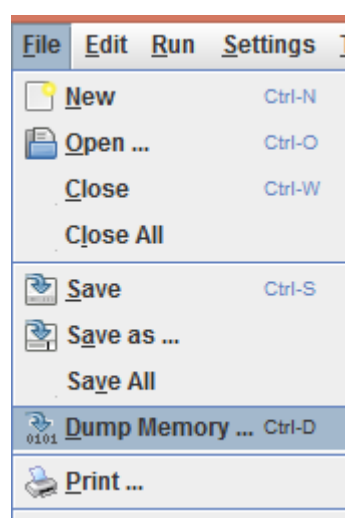


图 3-26 生成镜像操作图

然后在生成的文件头部加 v2.0 raw 即可供 logisim 使用。

直接调入到 logisim 中，启用时钟模拟：

中途的跑马灯部分截图图 3-27：

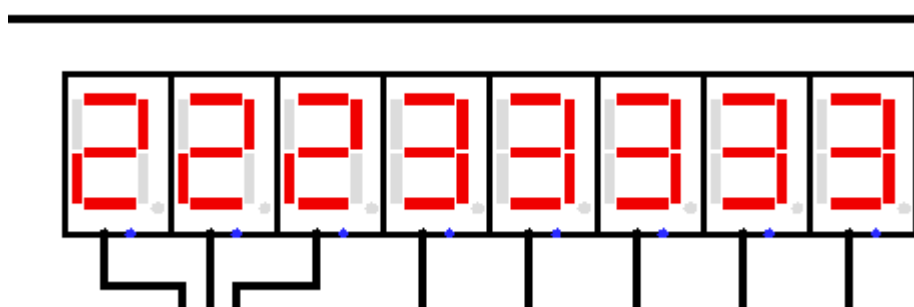


图 3-27 跑马灯部分效果截图





然后查看各类指令的统计条数：

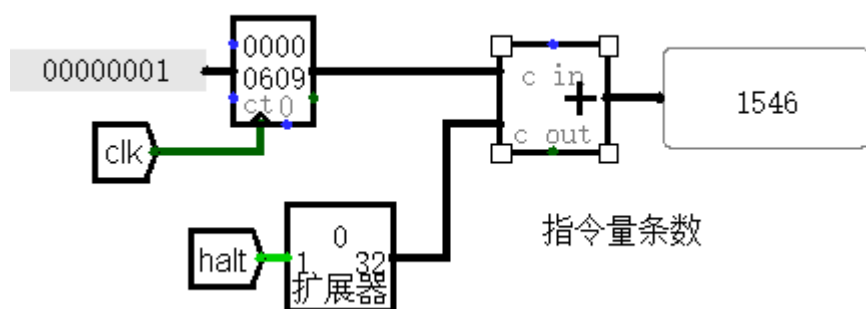


图 3-31 总指令条数图

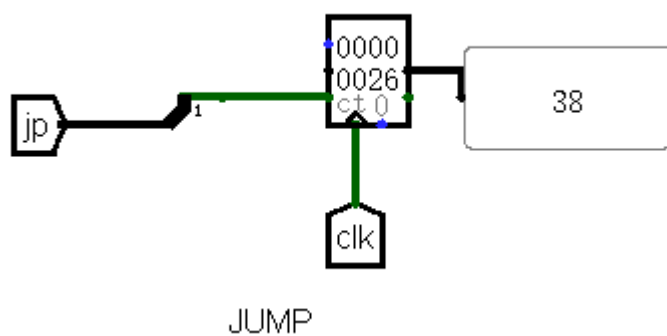


图 3-32 JUMP 型指令条数图

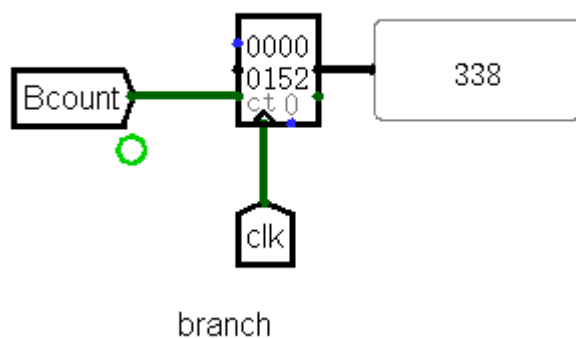


图 3-33 Branch 型指令条数图

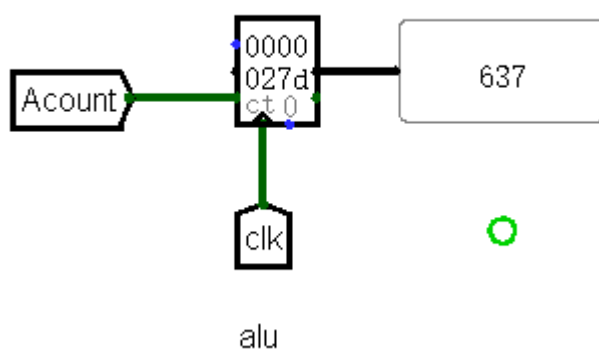


图 3-34 ALU 型指令条数图

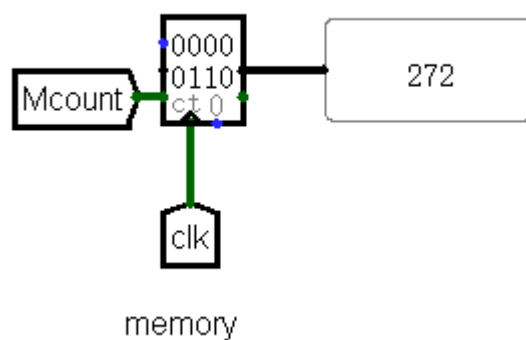


图 3-35 访存型指令条数图

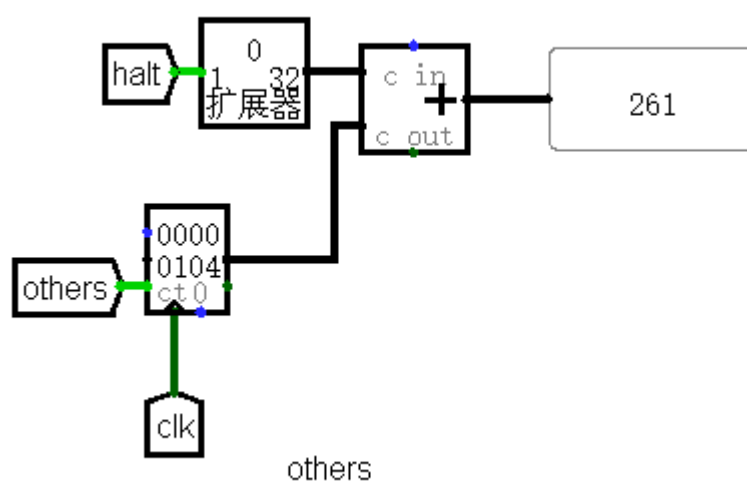


图 3-36 其他类型指令条数图

可以看出各类型的指令条数如上所示。

而实际指令条数可以通过将 benchmark 的汇编代码调入到 mars 中进行统计得出，统计结果为图 3-37：

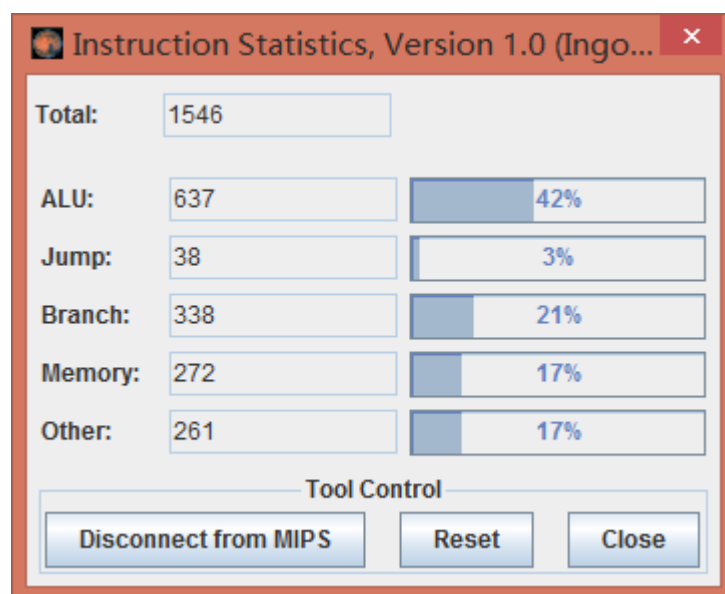


图 3-37 MARS 中指令条数查看图

可以看出，指令的计数和 mars 中的统计是相同的。

至此，指令条数相同，最终显示相同，内存中数据正确，本次实验完毕。

## 4 总结与心得

### 4.1 实验总结

本次综合实验共分为三个部分，分别是运算器的实现，寄存器组与数据通路的实现，单周期 24 指令 CPU 的实现，在实验中，虽然难度稍微偏大，但是收获颇丰，先列举如下：

- 1) 学会了对 logisim 模拟器的熟练使用，使用过程中体会到了随手保存中间结果的重要性。
- 2) 实验一首先完成了先行进位加法器的设计，设计过程中对课本中第三章的先行进位电路有了很好的理解和记忆，然后设计了扩展位（32 位）的并行加法电路，并且用自己设计的 32 位并行加法电路设计出来一个带检测溢出的运算器；实现了 CPU 中的核心部件—运算器的构建，较为顺利的完成了实验 1。
- 3) 实验二首先完成了寄存器组的设计，设计过程中对寄存器的存取特性，存取方式有了很好的理解；然后又进行了自动运算的数据通路的设计，设计过程中学会了如何进行地址的拼接，如何进行寄存器的循环使用，实现了加法的数据通路，为实验三做下了良好的铺垫。
- 4) 实验三完成了 CPU 的设计，设计中首先完成的是指令的译码和控制器的设计，译码器的设计通过分析电路，写表达式进行生成，虽说简单了许多但是还是需要细心，实现了译码器后进行了控制器的编写，控制器的编写让我体会到了每个指令是通过多条不同的“微指令”来实现的，一个指令的实现实际上就是处理好各条微指令之间的先后顺序；最后顶层模块的搭建时候，学习到了如何进行地址跳转以及各种选择，体会到了多路选择器在实验中的重要性。

### 4.2 实验心得

- 1) 本次组成原理实验的终极目标是在 logisim 中设计一个单周期的 CPU，在

# 华中科技大学课程实验报告

---

完成实验之前，完全不敢相信自己竟然在短短几周内实现了一个 CPU，虽然较为简陋，但是成就感还是非常强的。

- 2) 实验中的前两次实验是对第三次实验的铺垫，实验难度层层递进，类似于“搭积木”方式的实验进程，还是较为容易接受的。
- 3) 实验完成后，对 CPU，对寄存器组，对数据通路，对运算器等等的构造有了非常清晰又直观的感受，这对我们以后的学习，设计，甚至工作上都会有很大的帮助。
- 4) 本次实验是在大学三年中是需要动手能力较为强的，趣味性也非常强，通过自己连线，大方向有指导，小方向自己把握的方式进行设计，既不会跑偏，又可以将自己的想法，自己的思路放到里边，有很大的灵活性。
- 5) 虽然是连线的方式实现 CPU，但是使用的是软件的方式进行实现，避免了连线时候的线路短路或者芯片内部的问题，还记得去年做数字逻辑实验的时候，出现过很多的问题就是因为线内部断了而导致的。
- 6) 实验中对硬件的设计的多样性让我体会到了做硬件开发的可扩展性；但是随手一个错误就会导致各种问题的出现，所以在硬件开发的时候，一定要主要不能出现因为疏忽而发生的错误，这将会浪费很大的时间。
- 7) 硬件开发较为耗时间，如果可以将实验提前到学期中间做而不是学期末，应该可以做的更好。

总而言之，组成原理的实验和上学期的计算机系统基础的实验让我深刻的体会到了实验不仅仅是“编程”，也不仅仅是“纸上谈兵”，它有很大的实用性，对我们的日后生活会有很大的帮助，在今后，一定会继续好好做实验，在实验中学会的东西是忘不掉的，是终身受益的，此时此刻，我已经迫不及待的想去做组成原理的课程设计了。。。。

## 参考文献

- [1] DAVID A. PATTERSON(美). 计算机组成与设计硬件/软件接口(原书第4版). 北京: 机械工业出版社.
- [2] David Money Harris(美). 数字设计和计算机体系结构(第二版). 机械工业出版社
- [3] 秦磊华, 吴非, 莫正坤. 计算机组成原理. 北京: 清华大学出版社, 2011 年.
- [4] 袁春风编著. 计算机组成与系统结构. 北京: 清华大学出版社, 2011 年.
- [5] 张晨曦, 王志英. 计算机系统结构. 高等教育出版社, 2008 年.

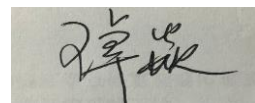
• 指导教师评定意见 •

## 一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字：



## 二、对课程实验的学术评语（教师填写）

## 三、对课程设计的评分（教师填写）

评分项目 (分值)	报告撰写 (30 分)	课设过程 (70 分)	最终评定 (100 分)
得分			

指导教师签字：\_\_\_\_\_

2017-01-14