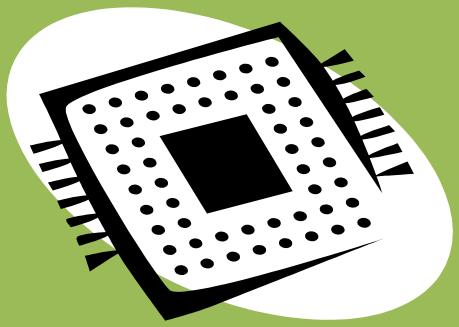
华中科技大学

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：	U201414803
姓　　名：	许荣仁
电　　话：	15629069226
邮　　件：	2267948095@qq.com
完成日期：	2017-01-13



计算机科学与技术学院

目 录

1 运算器实验

1.1 设计要求

1、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。

2、利用封装好的加法器以及 logisim 平台中现有运算部件（禁用系统自带的加法器，减法器）构建一个 32 位运算器，可支持算术加、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表 1.1 芯片引脚与功能描述 ，在主电路中详细测试自己封装的运算器，封装如图 1.1 所示，在报告中分析该运算器的优缺点。

表 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，具体功能见表 1.2
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
CF	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

表 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 Result2=0 (Set OF/CF)
0110	6	Result = X - Y Result2=0 (Set OF/CF)
0111	7	Result = X & Y Result2=0
1000	8	Result = X | Y Result2=0
1001	9	Result = X⊕Y Result2=0
1010	10	Result = ~(X |Y) Result2=0
1011	11	Result = (X < Y) ? 1 : 0 Signed Result2=0
1100	12	Result = (X < Y) ? 1 : 0 Unsigned Result2=0
1101	13	Result = Result2=0
1110	14	Result = Result2=0
1111	15	Result = Result2=0

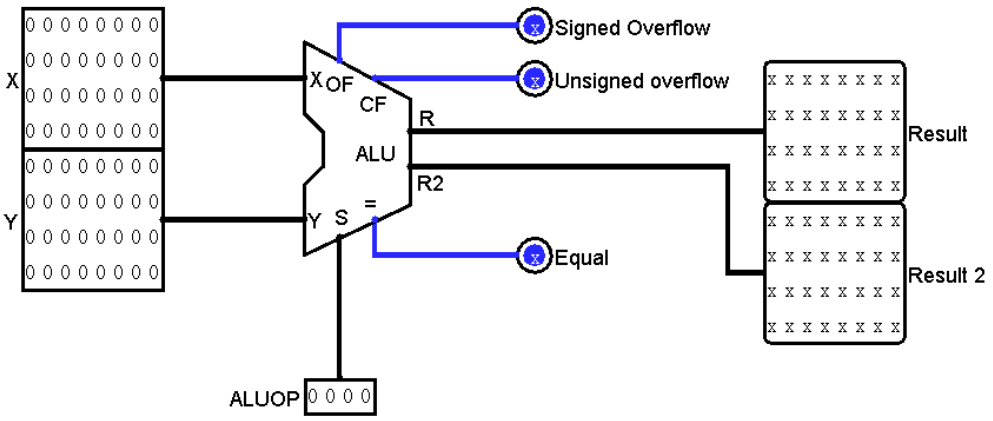


图 1.1 运算器封装示意图

1.2 方案设计

1.2.1 快速加法器

快速加法器不能使用简单的串行方式，需要先构造产生各位的进位生成和传递电路，再构造先行进位电路 74182，然后利用 74182 构造可以先行进位的 4 位运算单元 74181，再利用多个 74181（每个芯片能算 4 位，计算 32 位需要 8 个芯片）和 74182（每组的 G P 信号需要再经过芯片的处理后用作其它芯片的进位输入，需要 3 个芯片）构造 32 位快速加法器。

1.2.2 运算器封装

运算器可以根据 op 信号的不同实现对输入数据的不同操作，产生不同的输出，实现的思路是先把所有的操作同时完成，再用多路选择器，根据 op 信号选择不同的结果输出，多路选择器的选择信号是 4 位，根据所给的表，0000 是逻辑左移对应的数据输入端第 0 个就连接逻辑左移运算后的结果，0001 是算数右移，对应的数据输入端第 1 个就连接算数右移运算的结果，依次连接。

大部分的运算都可以直接一步通过单个元件完成，只有移位运算需要先取 Y 的低 5 位，减法需要先对 Y 取反加 1 变成加法运算，比较运算后的结果需要经过位扩展器扩展成 32 位。

多路选择器的连接。封装的运算器有 5 部分输出，但有一个 equal 是直接通过比较器就可以给出，不需要进行多路选择，所以需要 4 个多路选择器，分别是 result，result2，OF 和 UOF，其中 result 需要与所有的运算器相连，result2 仅有乘除法用到，只与乘除法运算器相连，OF 和 UOF 只与加减法相连，所有多路选择器不连接运算器的位都置为 0。

最后用隧道把对应的端口与题目给定的端口连接起来。

1.2.3 总体结构图

32 位加法器的总体结构图如图 1.2 所示，ALU 的总体结构图如图 1.3 所示：

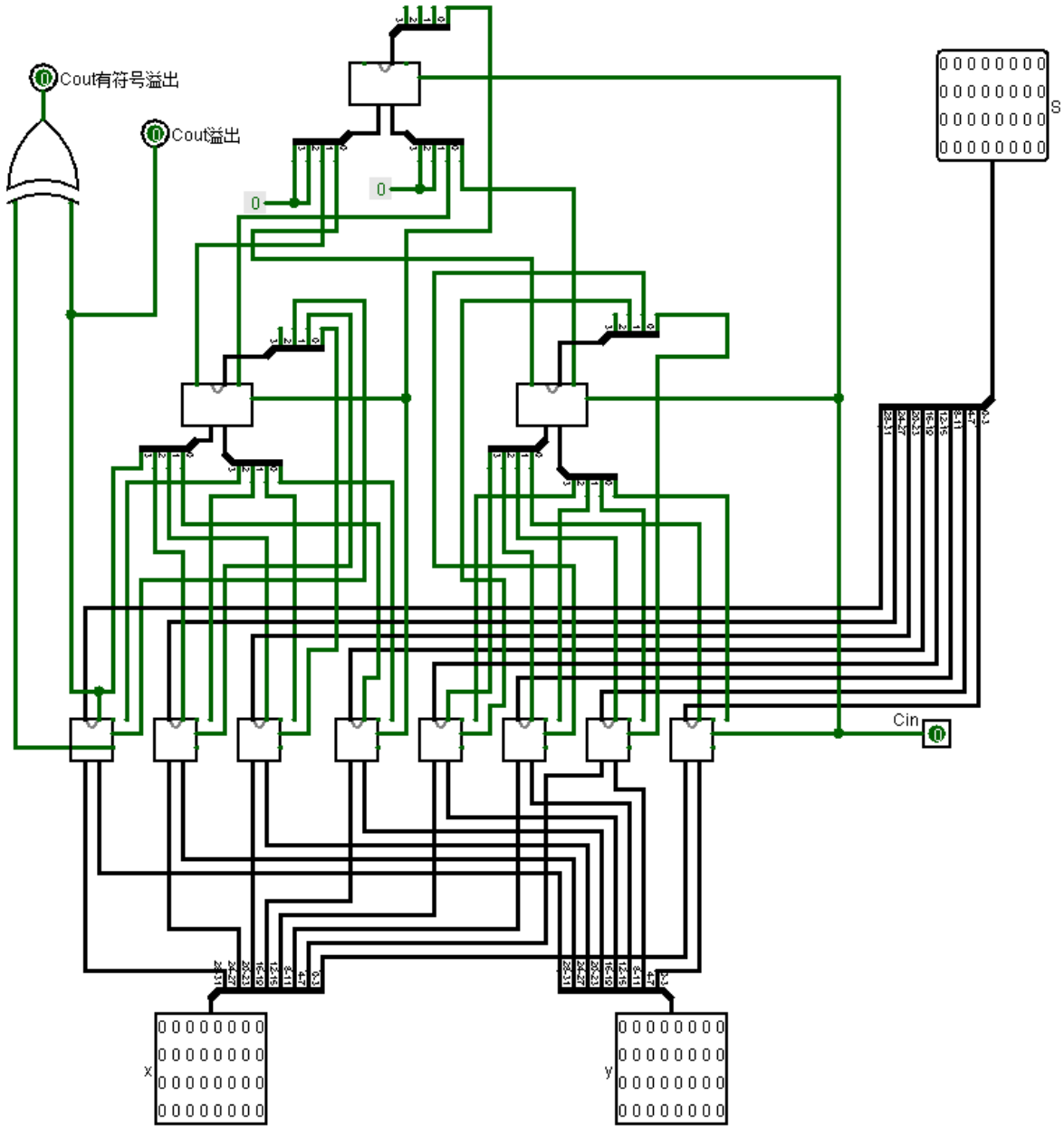


图 1.2 32 位加法器总体结构图

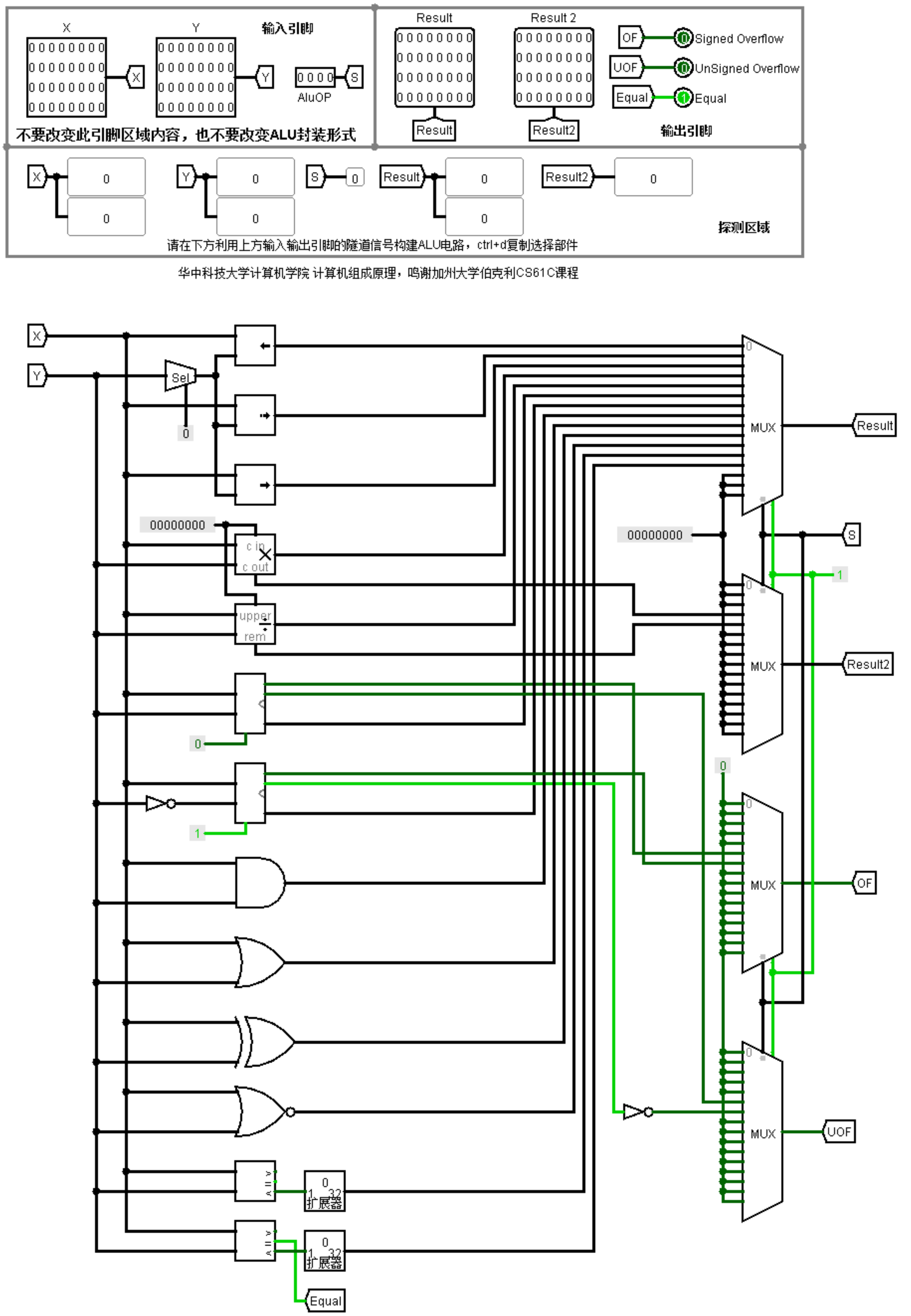


图 1.3 ALU 总体结构图

1.3 实验步骤

（1） GP 生成电路，根据四位输入与进位输入产生 GP 信号；

（2） 74182 电路，根据上一部的 GP 信号产生 G*P*信号；

（3） 74181 电路，利用 74182 芯片构建 4 位快速加法器；

（4） 32 位加法器，利用前面完成的 74181 芯片和 74182 芯片，完成 32 位快速加法器

（5） 完整 ALU，利用上述完成的加法器和 logisim 提供的其他部件，根据要求连接完整电路。

（6） 测试。

1.4 故障与调试

1.4.1 减法运算错误

故障现象：减法运算的结果不正确。

故障分析：减法变加法的过程中只把 Y 取了反，忘记在进位位加 1。

解决方案：把减法运算的进位位置为 1.

1.5 测试与分析

测试 ALU 共使用了 21 组用例，具体测试用例见表 1.3 。

表 1.3 各个输出测试用例

#	A	B	OP	运算	OF	UOF	Result	Result2
1	1052672	134742016	0101	加	○	○	135794688	0
2	3221229568	1211629568	0101	加	○	●	137891840	0
3	1073745920	2089296896	0101	加	●	○	3163042816	0
4	2147487744	2151153664	0101	加	●	●	3674112	0
5	134221824	76030976	0110	减	○	○	58190848	0
6	1073745920	2089296896	0110	减	○	●	3279416320	0
7	3221229568	1149772800	0110	减	●	○	2071456768	0
8	1207963648	2223514624	0110	减	●	●	3279416320	0
9	98304	1680384	0011	乘			1979711488	38
10	32768	41984	0011	乘			1375731712	0
11	1081344	41984	0100	除			25	31744
12	2574336	41984	0100	除			61	0
13	2884560	526435	0000	<<			23076480	0
14	2150368208	526435	0001	>>>			4026892410	0
15	2150368208	526435	0010	>>			268796026	0
16	2150368208	526435	0111	按位与			524352	0
17	2150368208	526435	1000	按位或			2150370291	0
18	2150368208	526435	1001	按位⊕			2149845939	0
19	2150368208	526435	1010	按位~(|)			2144597004	0
20	2147746824	3147776	1011	符比较			1	0
21	2147746824	3147776	1100	U 比较			0	0

测试了四组加法，四组减法，两组乘法，两组除法，一组逻辑左移，一组算术右移，一组逻辑右移，一组按位与，一组按位或，一组按位异或，一组按位或非，一组有符号比较，一组无符号比较，测试结果如图 1.4 到图 1.24 所示：

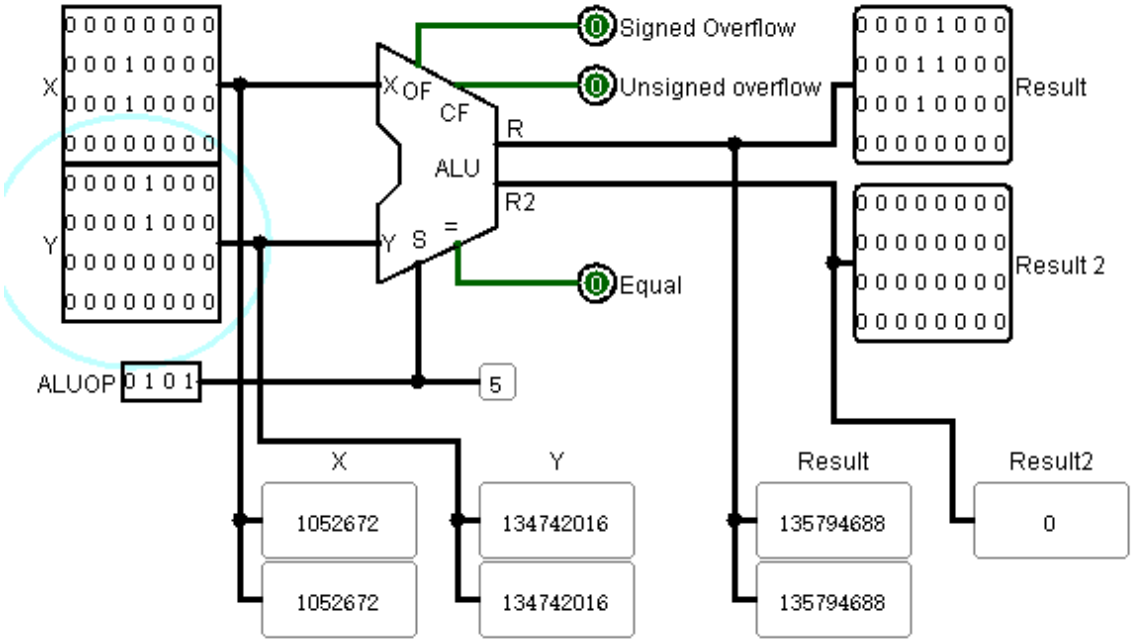


图 1.4 加法测试用例 1

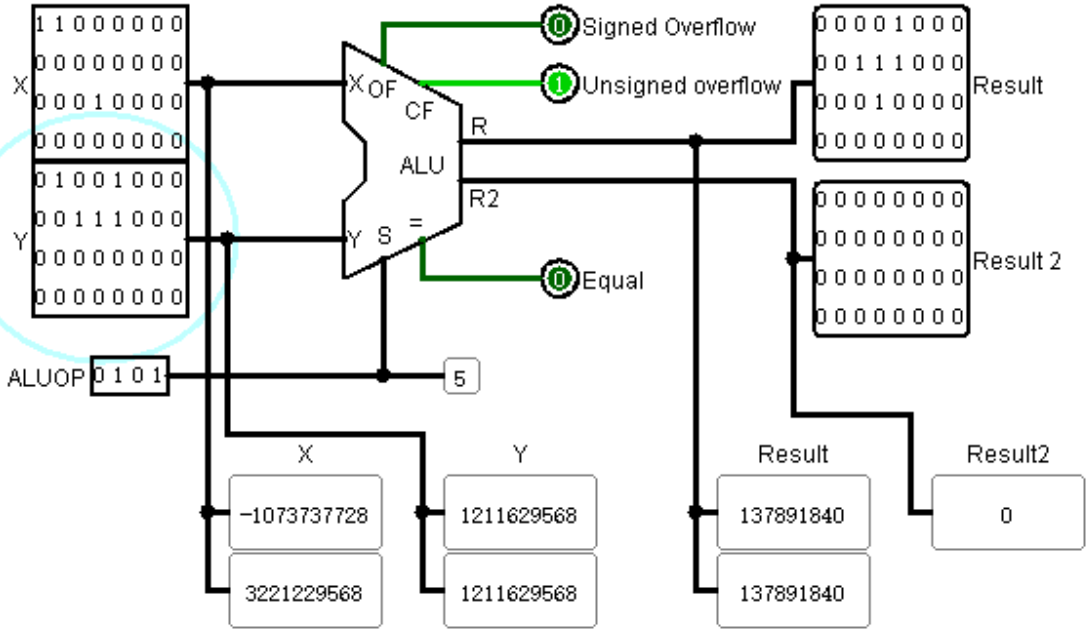


图 1.5 加法测试用例

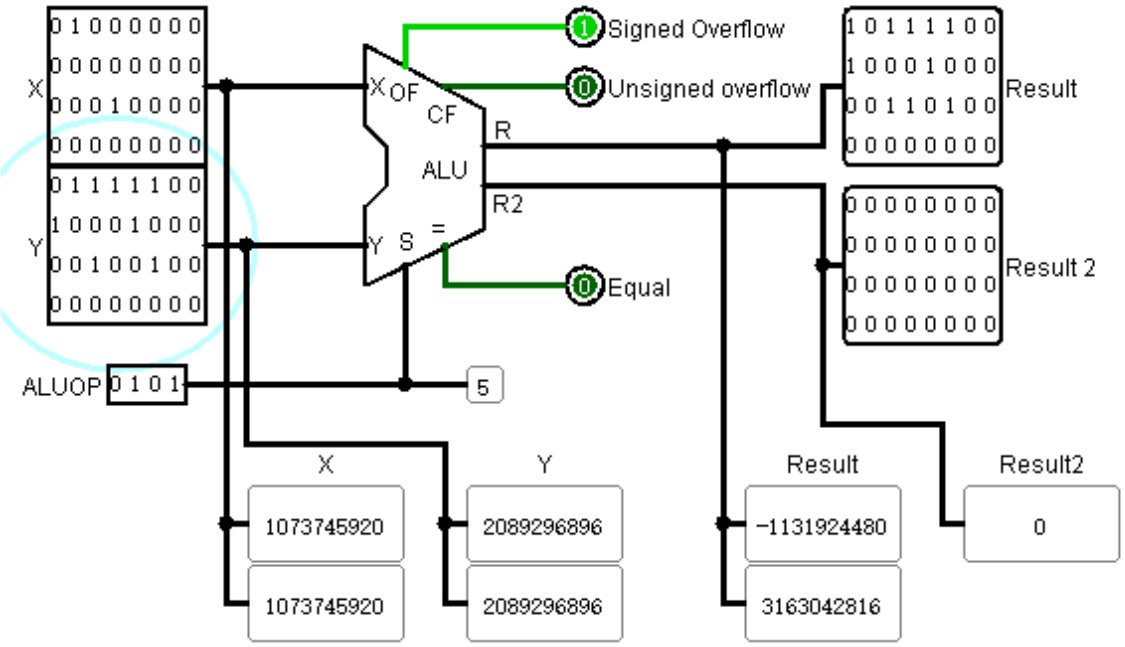


图 1.6 加法测试用例

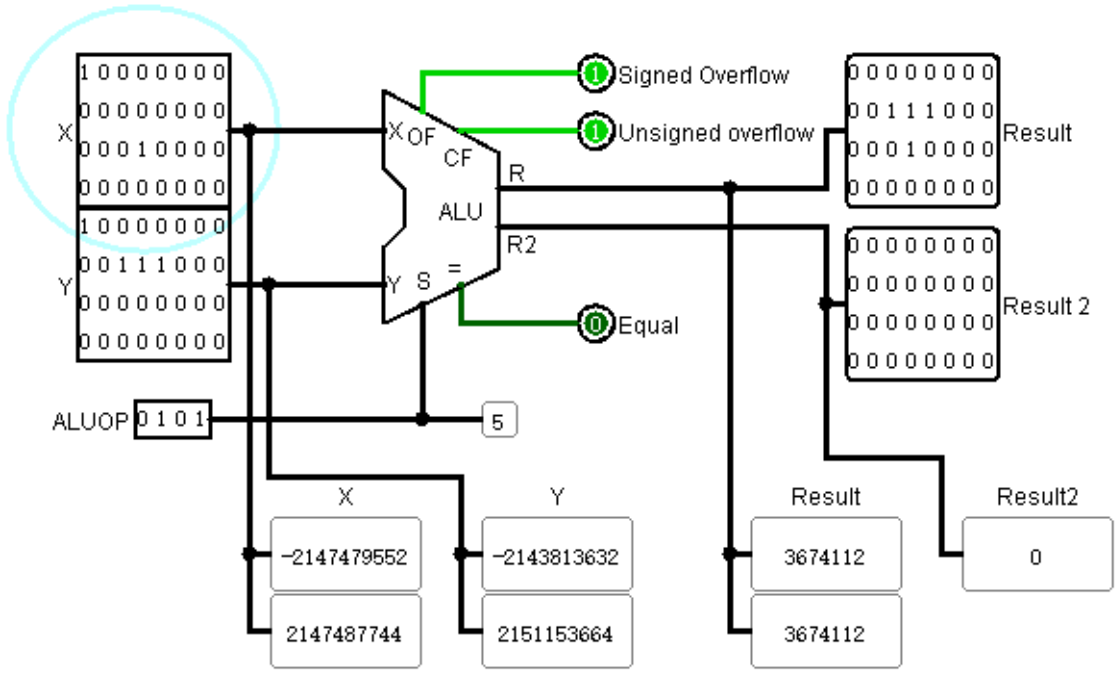


图 1.7 加法测试用例

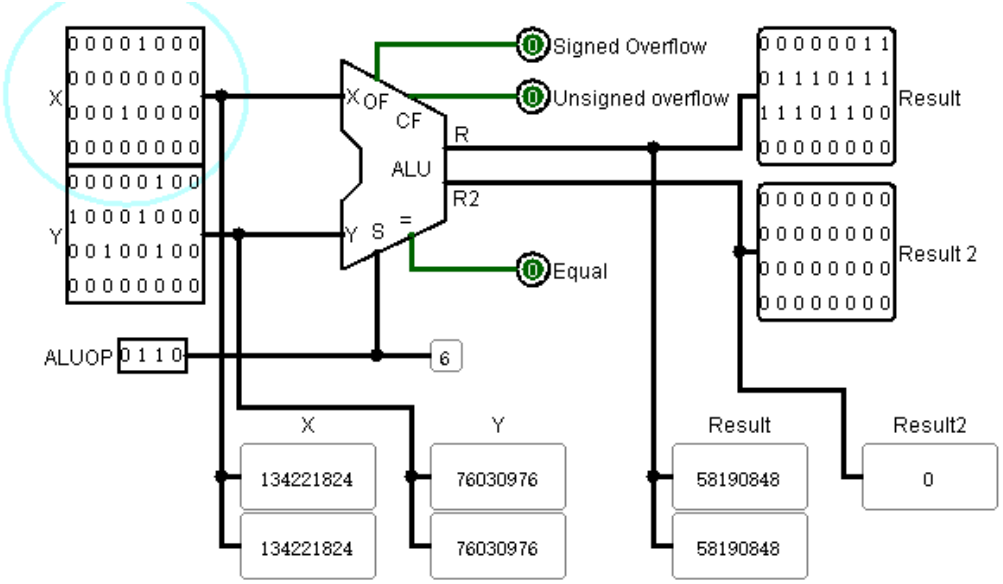


图 1.8 减法测试用例

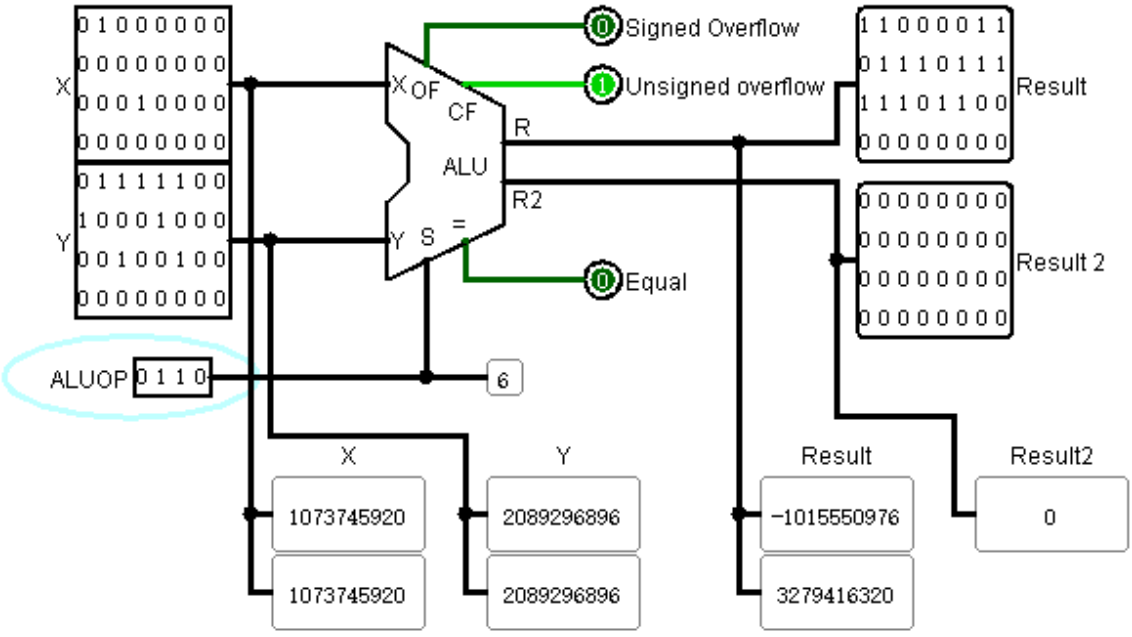


图 1.9 减法测试用例

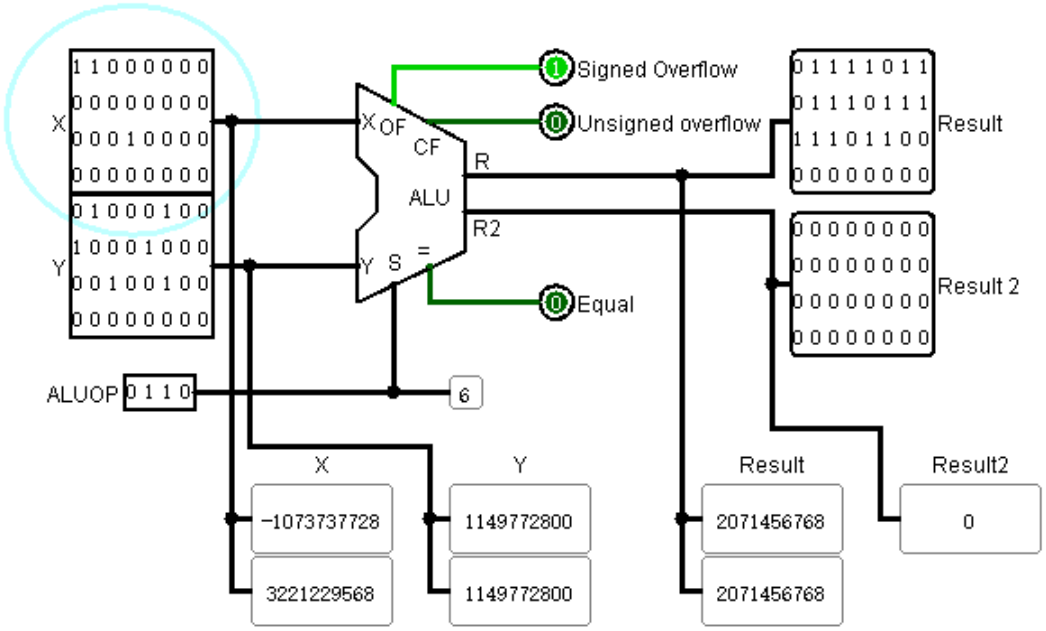


图 1.10 测试用例

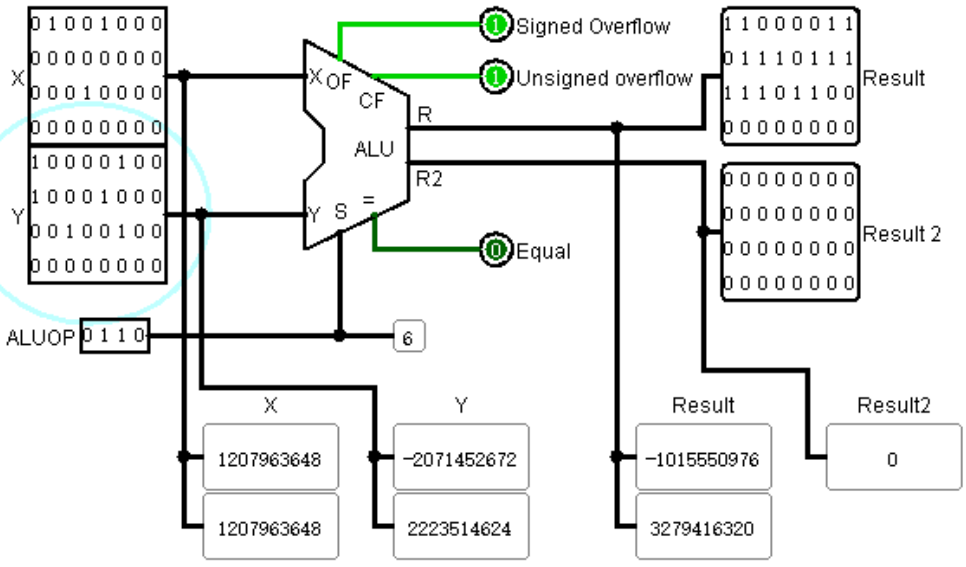


图 1.11 减法测试用例

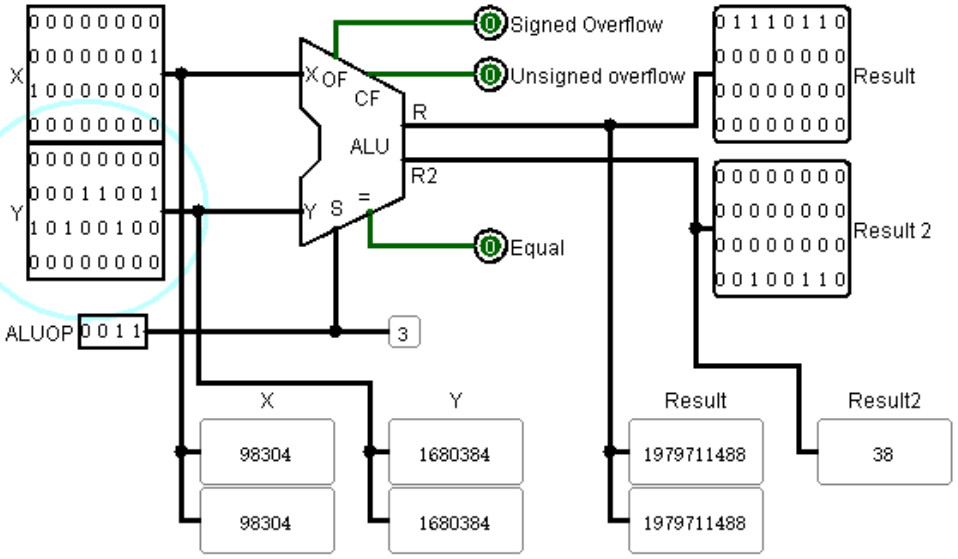


图 1.12 乘法测试用例

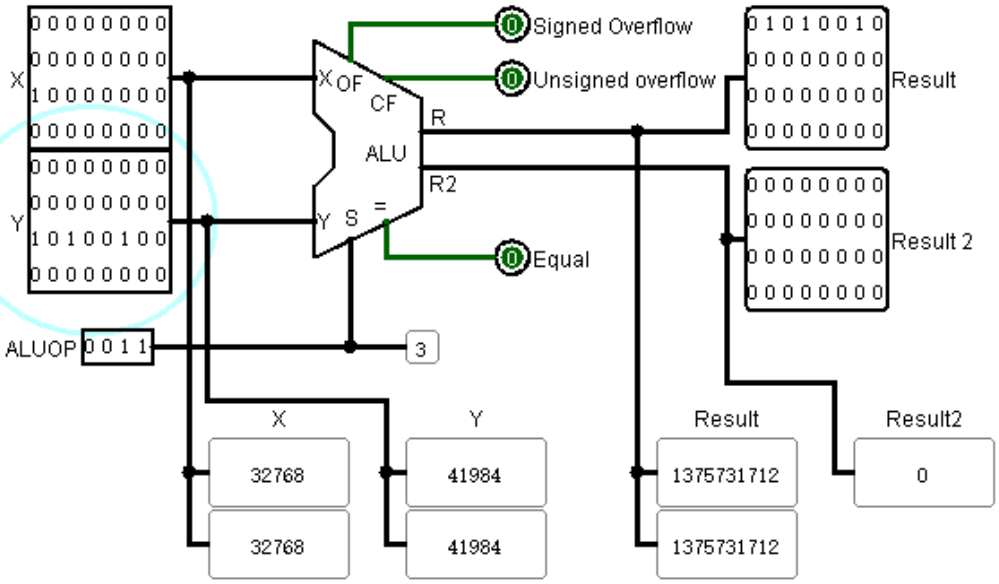


图 1.13 乘法测试用例

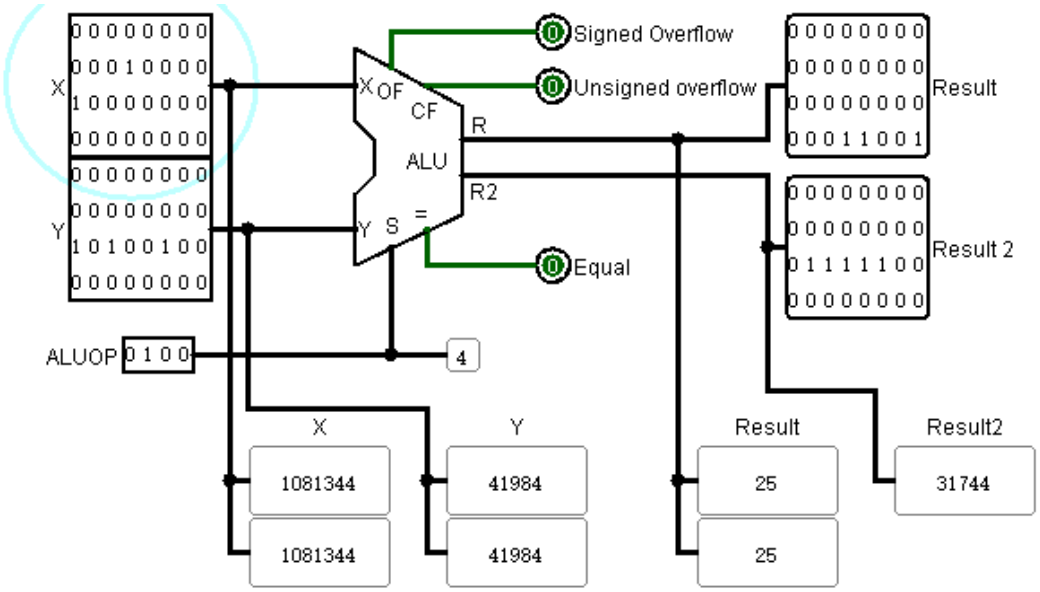


图 1.14 除法测试用例

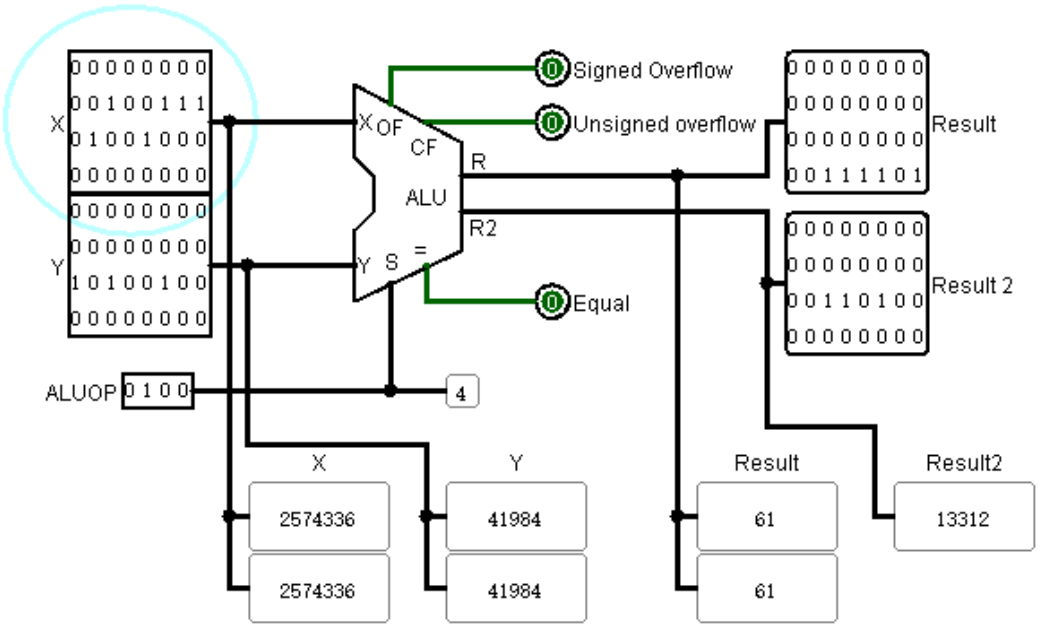


图 1.15 除法测试用例

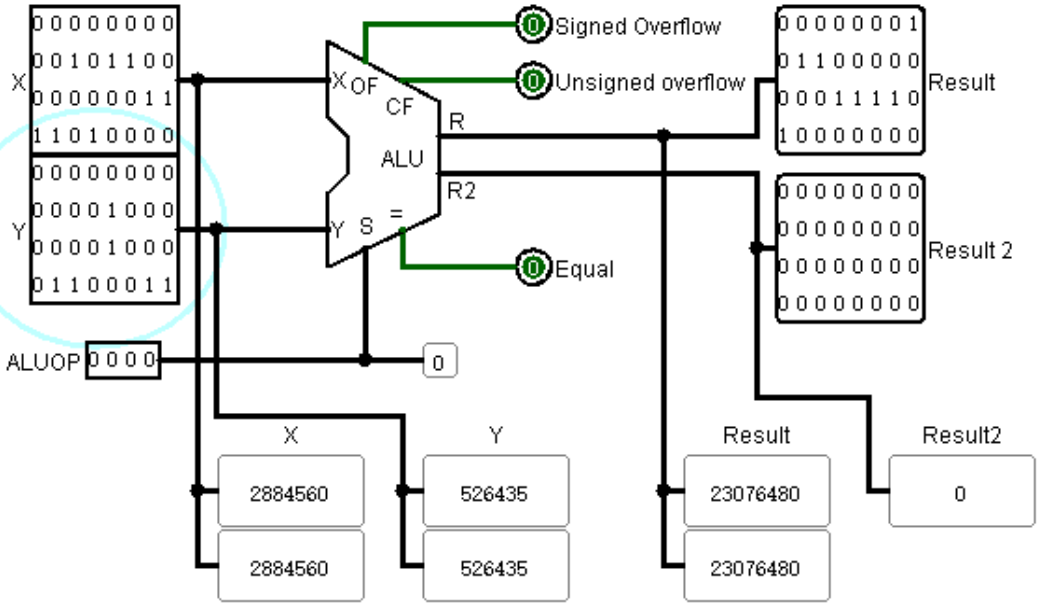


图 1.16 逻辑左移测试用例

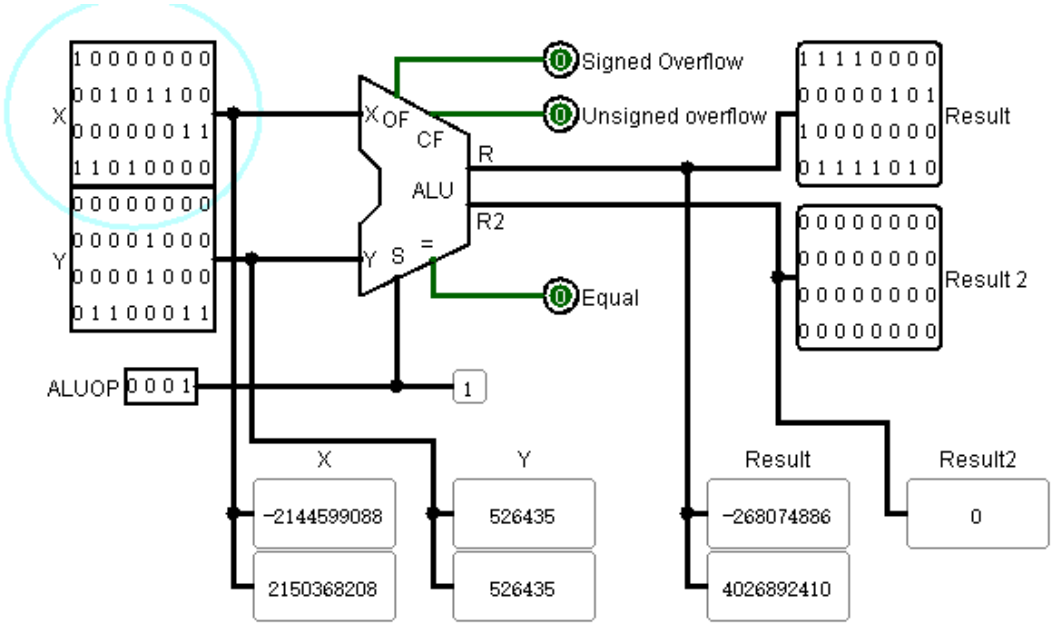


图 1.17 算术右移测试用例

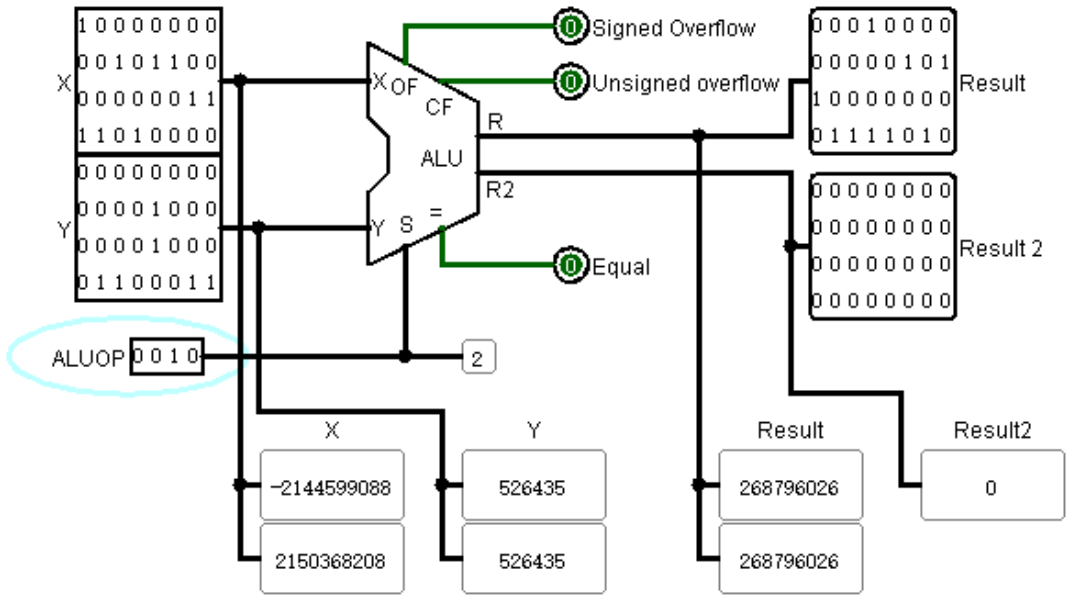


图 1.18 逻辑右移测试用例

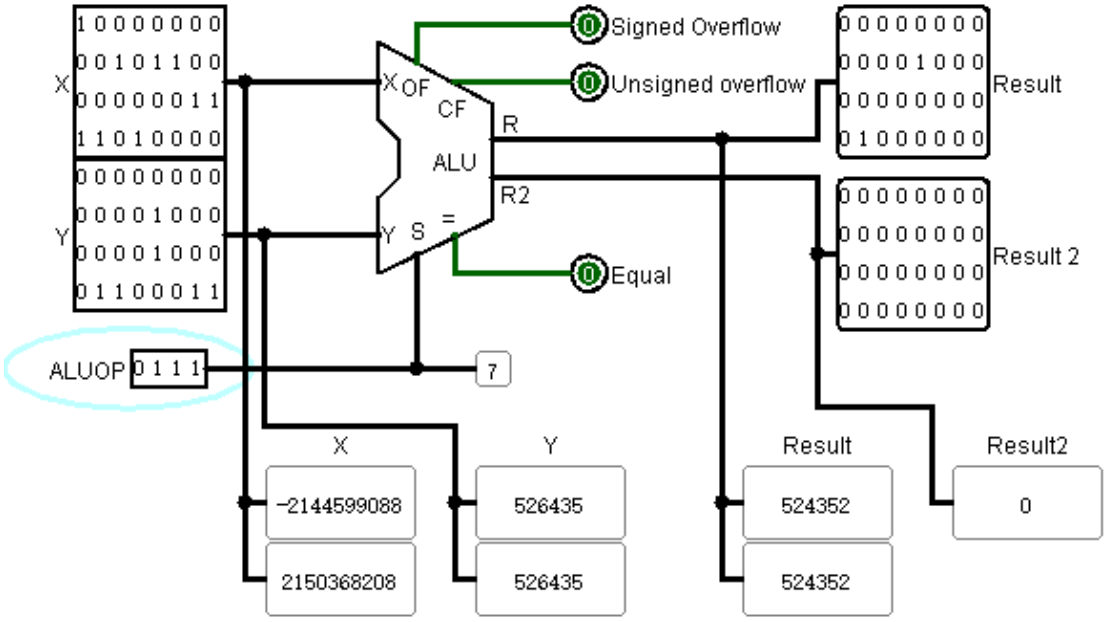


图 1.19 按位与测试用例

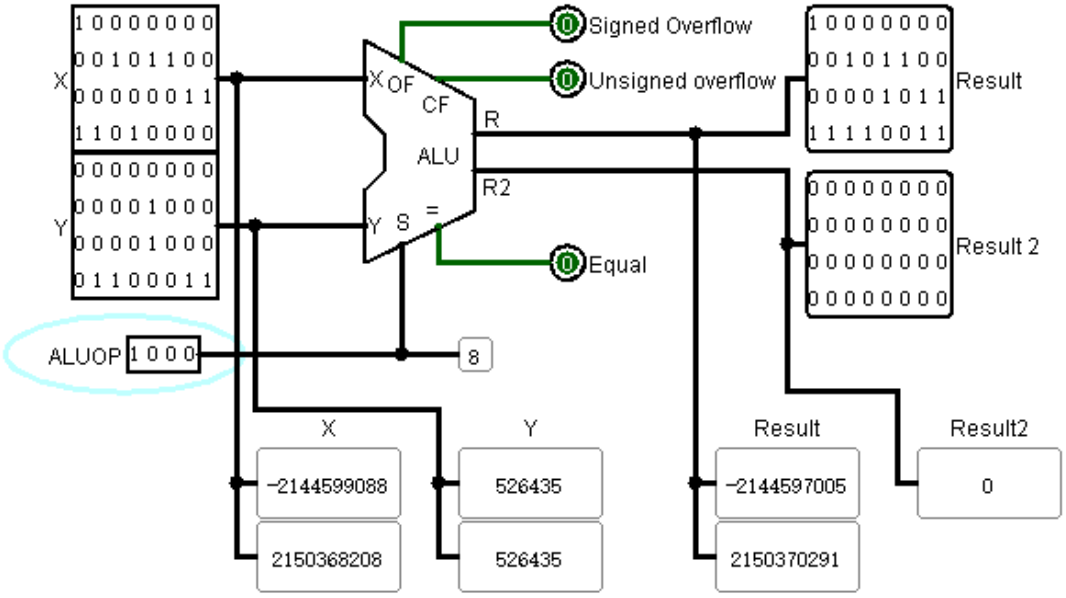


图 1.20 按位或测试用例

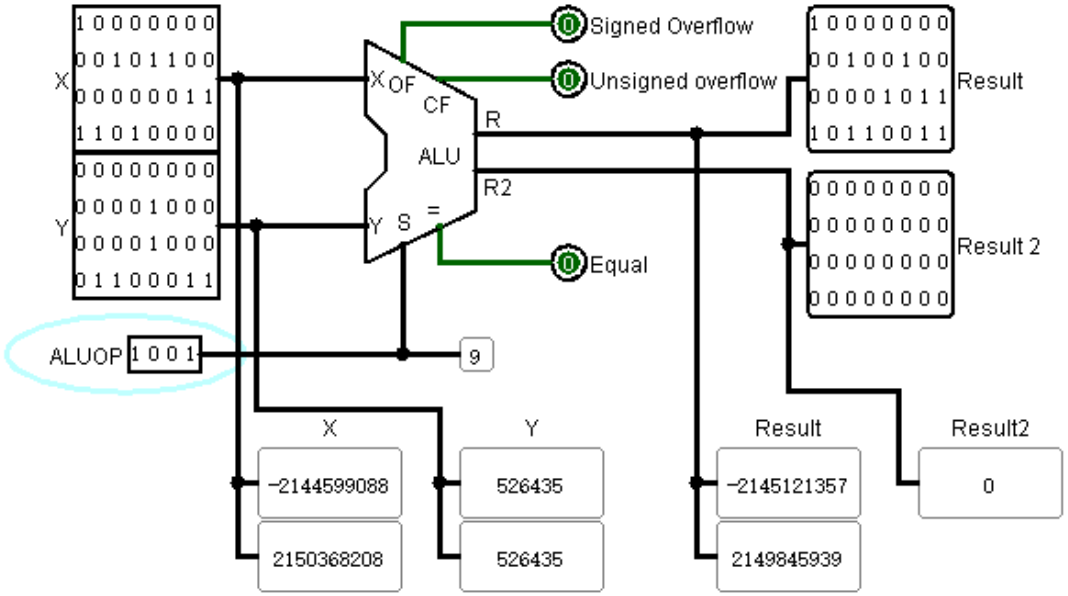


图 1.21 按位异或测试用例

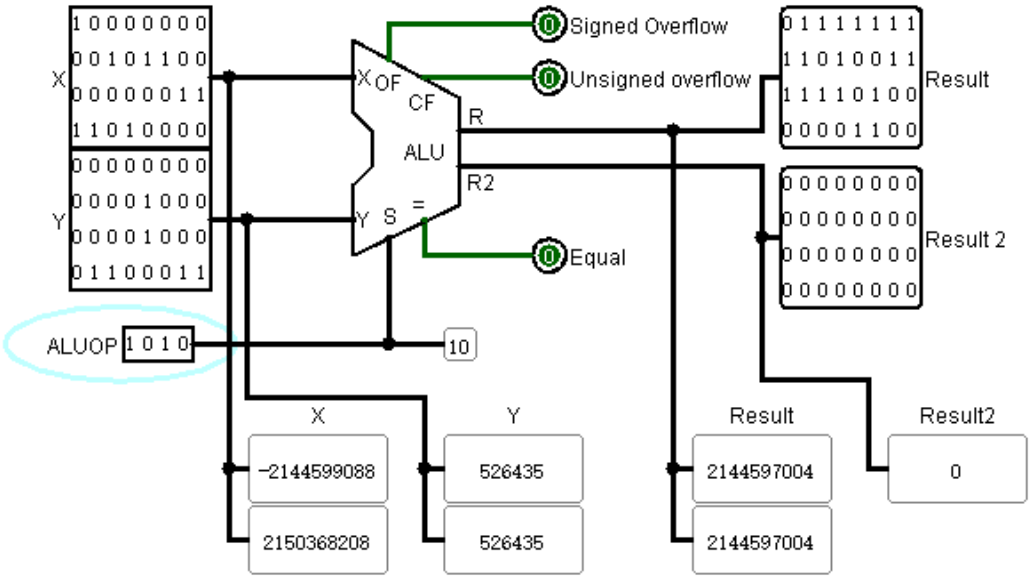


图 1.22 按位或非测试用例

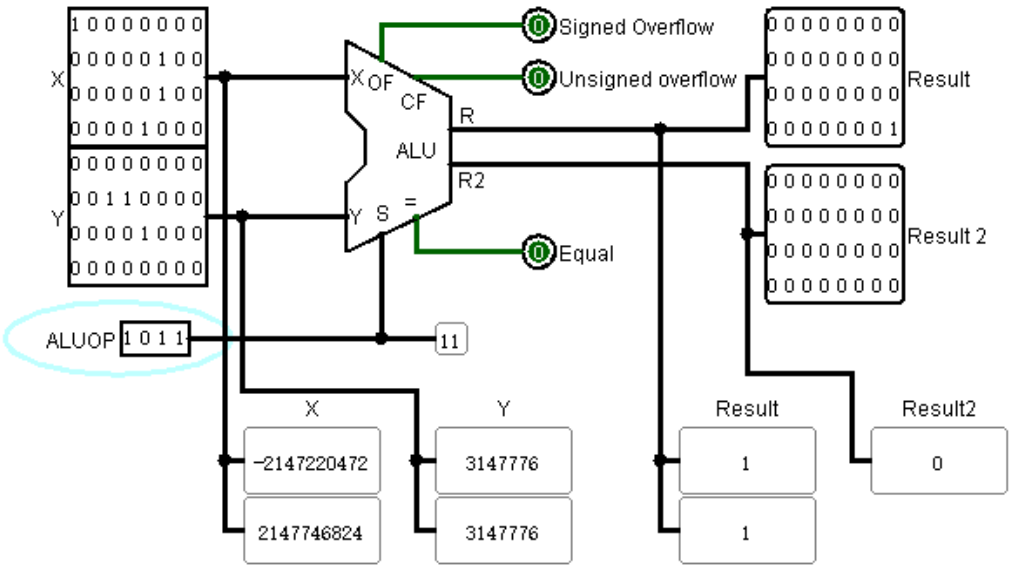


图 1.23 符号比较测试用例

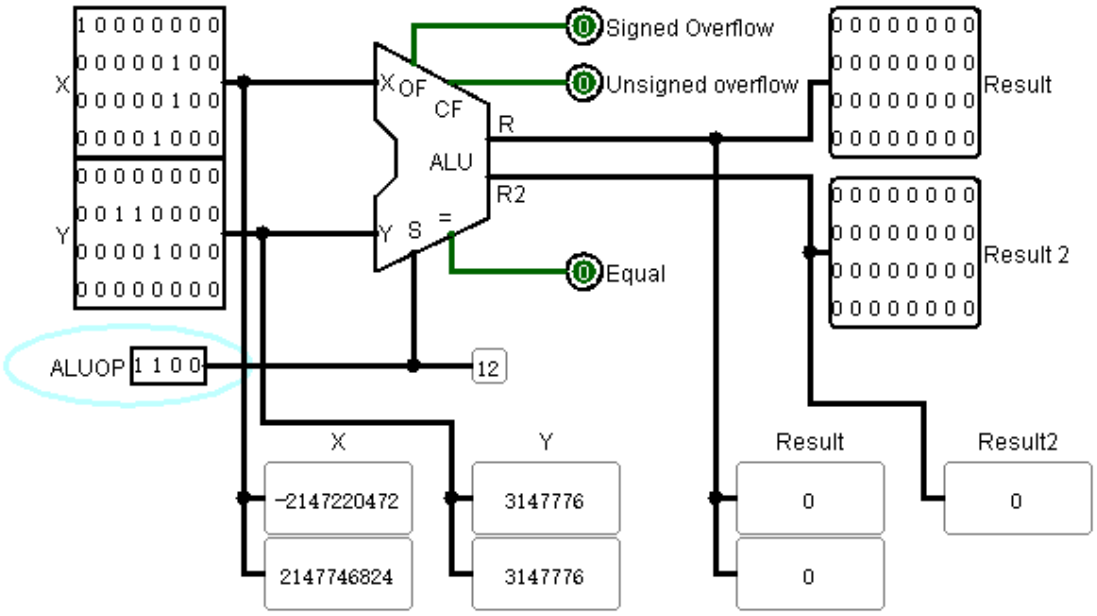


图 1.24 无符号比较测试用例

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，其中 0 号寄存器的值恒为零，其具体功能如下表 2.1，芯片引脚如图 2.1 所示。

表 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 上调沿将 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

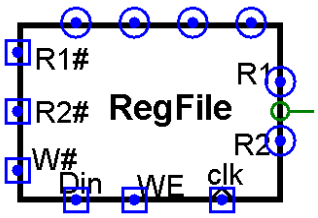


图 2.1 芯片封装引脚

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。

主电路最上面一行请将所有关键点的值用探测和隧道方式结合引出，用 10 进制方式显示，便于检查，运算器结果直接用 16 进制数码管显示。

2.2 方案设计

2.2.1 RegFile 模块

RegFile 理论上应包含 32 个 32 位寄存器，但由于第一个恒为零，所以实际上需要 31 个寄存器，第 0 个用 32 位的常量 0 代替；

每个寄存器都需要读或者写，但读哪两个是由输入信号 R1#和 R2#决定的，所以需要两个多路选择器选择要输出的数据，多路选择器的输入数据端都连接所有的寄存器输出端，选择端分别连接 R1#和 R2#，输出端分别连 reg1 和 reg2；写哪一个寄存器是有 RW#决定的，所以要用两个解复用器把要写入的数据和写使能信号传送给相应的寄存器；

16、17、18 和 31 号寄存器输出直接连接到对应的端口（$s0, $s1, $s2,$ra）即可。

2.2.2 自动运算模块

完成有一定次数的自动运算，需要用计数器来控制运算终止，当计算了 32 次后，应该产生一个停止信号，使电路停止运算，比较简单的方法是用或门屏蔽掉时钟信号；

运算时需要先从 RAM 中取出数据，计算后再写入到 RAM 中，这两部要在一

个时钟周期内完成，所以要错开两个操作，分别使用时钟的上升沿和下降沿触发，并且写的地址和读的地址不同，用二路选择器根据时钟所处的状态选择使用哪一个地址；

寄存器也要进行读和写操作，因为只是暂时存储运算结果，所以只需要使用寄存器文件中的一个寄存器即可，也就是把读和写地址设为固定的一个，数据输入端连接 ALU 的计算输出端；

ALU 运算部件，X 端连接寄存器文件的输出，但是第一次加的时候，寄存器中没有内容，需要直接将 0 与 RAM 中第一个数据相加，所以再加一个二路选择器，仅当第一次加的时候选 0，其他时候都选择寄存器的输出；Y 端连接 RAM 的数据输出端，为了避免 RAM 因地址改变造成的输出数据不必要的变化传递到 Y，在两者之间加一个寄存器进行缓冲，随时钟更新数据；加法运算，ALU 的 op 端固定为 5，不需要溢出信息；

中间数据的查看，用隧道和探测器查看实时的计数、RAM 地址、RAM 数据等信息，每一步计算的结果通过 16 进制数码管显示，数据是 32 位，每个数码管可以显示 4 位宽的数据，所以需要 8 个数码管，使用分线器分别连接到对应的位。

2.2.3 总体结构图

寄存器文件的总体结构如图 2.2 所示，可以看出是包含 31 个寄存器，两个多路选择器，两个解复用器，还有多个隧道。

自动运算电路的总体结构如图 2.3 所示，可以看出有一个寄存器组文件，一个 RAM，8 个 16 进制数码管，一个 ALU，多个多路选择器，多个基本逻辑门，他们共同 构建成自动运算电路。

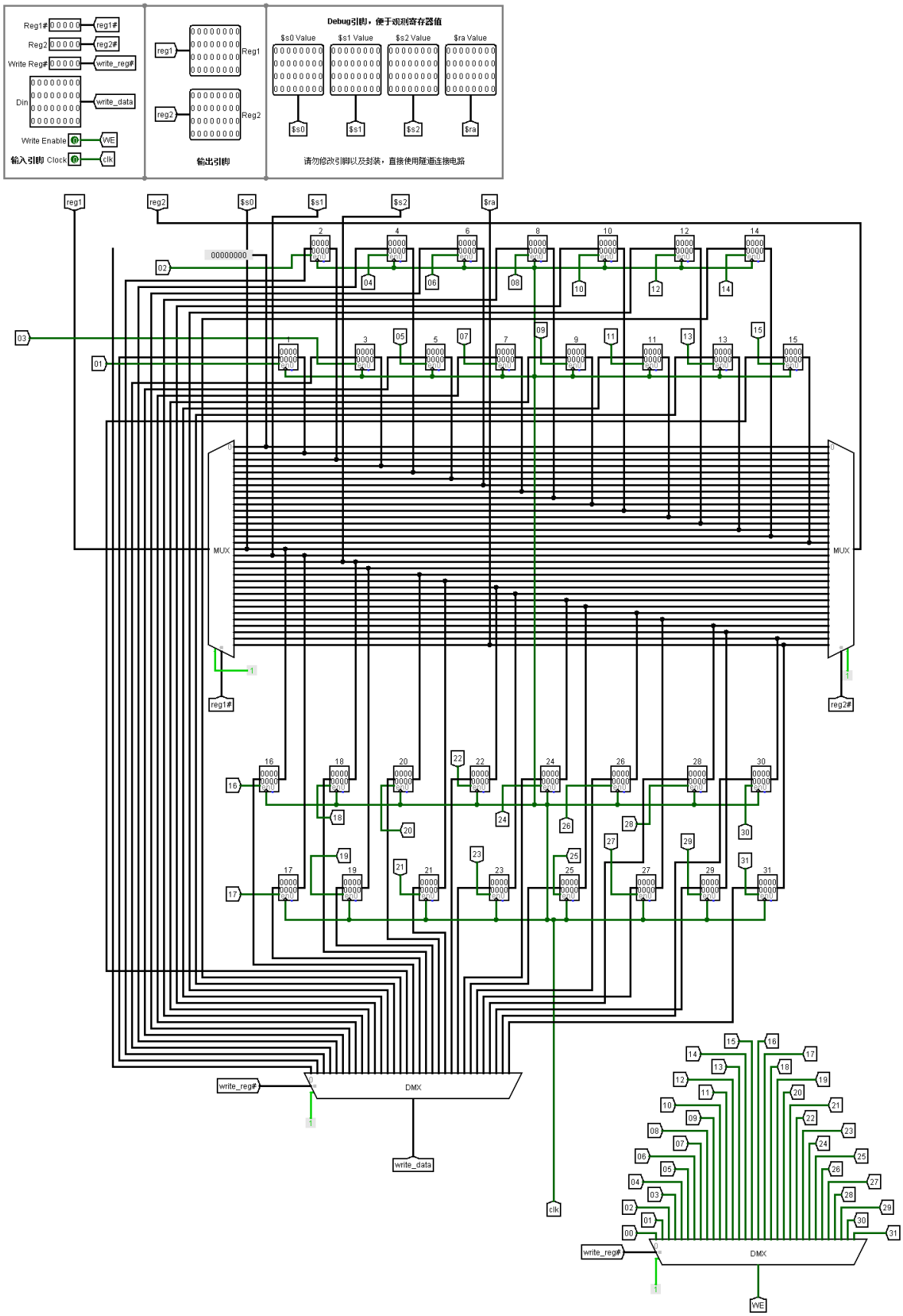


图 2.2 寄存器文件总体结构

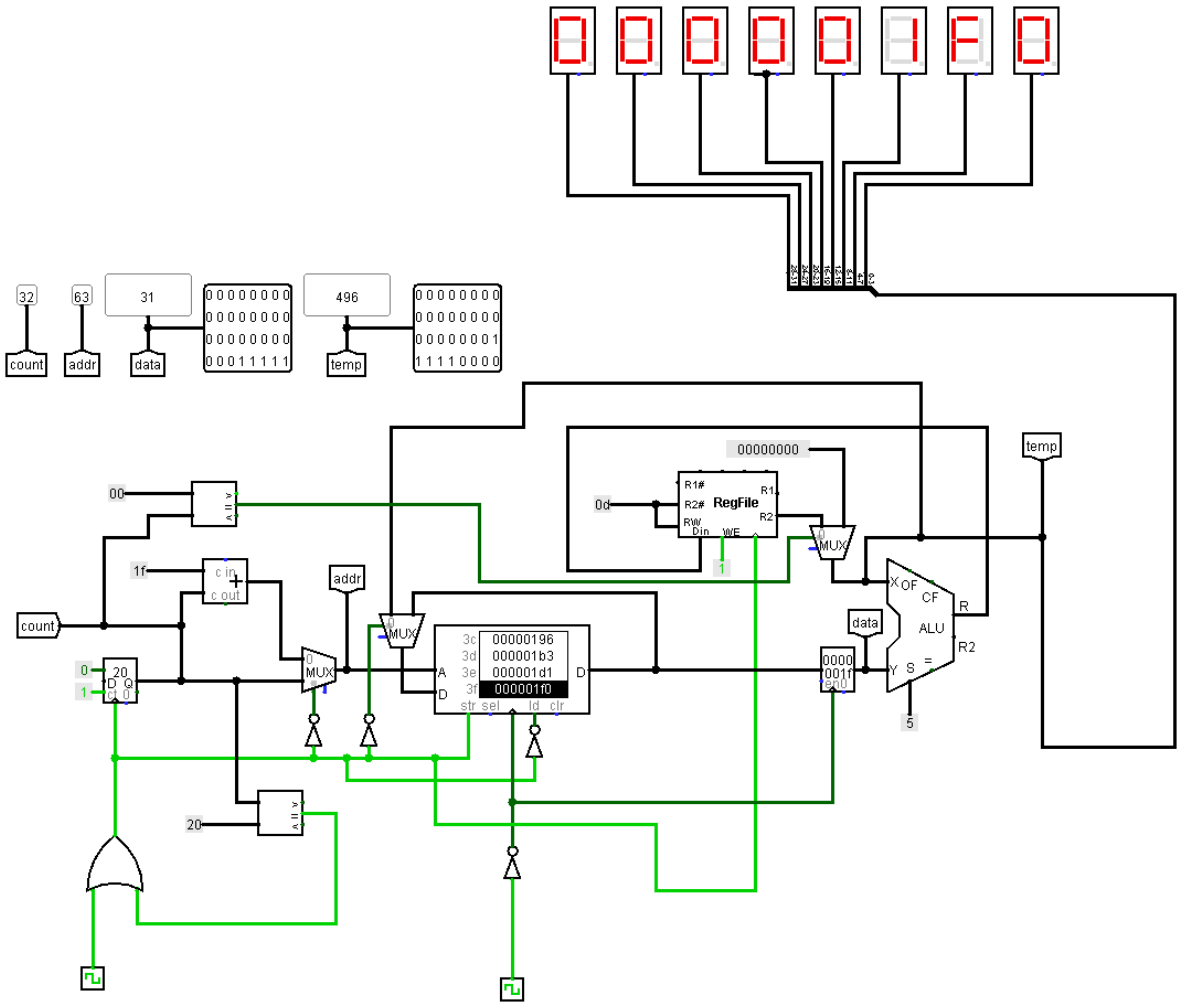


图 2.3 自动运算电路总体结构

2.3 实验步骤

（1） 寄存器文件，把 31 个寄存器和一个常量按照合适的位置摆放好，将两个多路选择器放，相同的位连起来，把寄存器的输出连接到对应的线上，再把四个需要输出的寄存器连接到对应的隧道，寄存器文件输出就可以了；写入要用两个解复用器，其一把写使能信号加载到对应的寄存器上，其二把要写入的数据加载到相应寄存器的数据输入端，并连接时钟。

（2） 自动运算电路，根据上述设计思路连接好时钟源，寄存器文件，RAM 模

块，ALU，16 进制数码管以及其他需要的控制部件与选择部件等。

（3） 运行测试修改

2.4 故障与调试

2.4.1 寄存器写数据问题

故障现象： 在每进行一次寄存器写数据的操作后，除了这一次写的寄存器外，其他寄存器都会被清零，如图 2.4 所示，在写入 17 号寄存器的时候，16 号寄存器的内容被清除了。

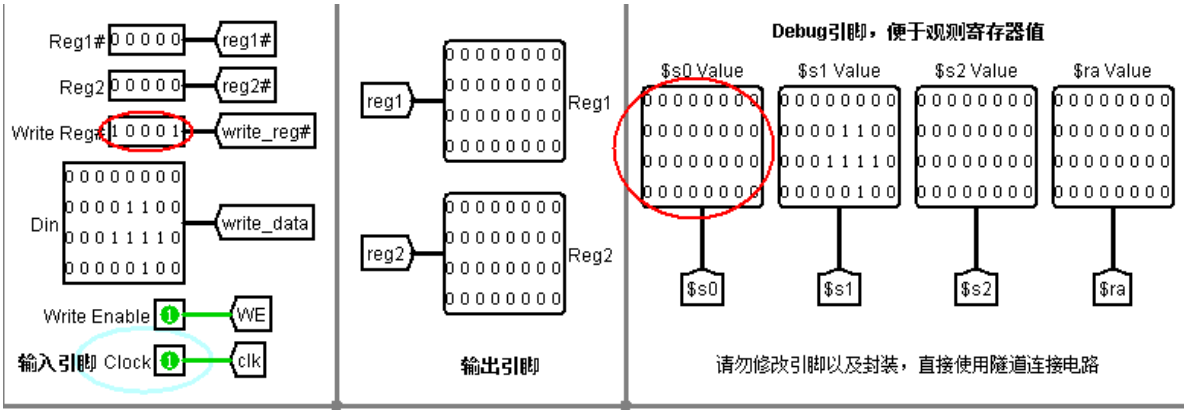


图 2.4 寄存器写入故障

原因分析： 使用解复用器控制写入时，只把数据进行了选择性输出，使能信号没有对应到需要写的寄存器上，而是把所有寄存器的使能端都置为了 1，导致需要写数据的寄存器能够写入，但其他寄存器都会被写入 0，抹去了原有数据。

解决方案： 把使能信号通过解复用器连接到寄存器的使能端，让寄存器只有在被选择时才能够写入数据。

2.4.2 自动运算电路保存运算结果时会从第 0x21 个 RAM 单元开始保存

故障现象：读地址再加 0x20 当作写地址时，保存计算的结果会从第 0x21 个 RAM 单元开始保存，跳过了第 0x20 个单元，并且导致最后一次的结果保存到了 0x0 号单元中，覆盖了原有数据 0，如图 2.5 所示。

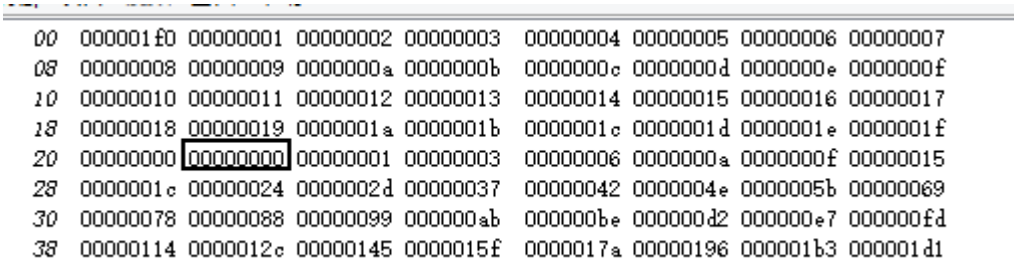


图 2.5 RAM 写入故障

原因分析：当第一个时钟上跳后，计数器已经加了 1，再加 0x20 就是 0x21，所以写入的地址会跳过 0x20，从 0x21 开始，依次后推一个单元。

解决方案：把加的 0x20 改为 0x19，就可以抵消不同步带来的多加 1 问题，使写入地址从 0x20 开始增加。

2.4.3 ALU 的 Y 端直接连接到 RAM 输出端会有错误状态

故障现象：当把 ALU 的 Y 端直接连接到 RAM 模块的输出端时，在运算程中会出现未知状态，导致 ALU 输出短暂的错误状态，虽然没有影响最后的结果，但是存在问题，如图 2.6 所示，ALU 的 Y 输入端未知，result 端错误。

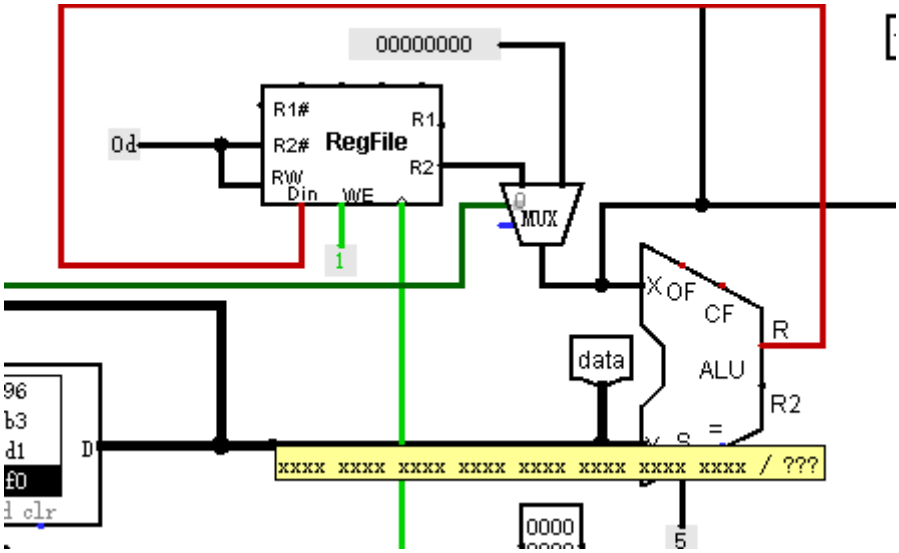


图 2.6 错误状态故障

原因分析：RAM 的输出端会实时显示所给地址的数据信息，由于在一个周期内 RAM 要同时完成读和写的操作，也就是会有两个不同的地址在一个周期内先后给

出，就会导致 RAM 输出端会先后输出两个地址的信息，而后给出的地址是需要保存计算结果的地方，不能把值传递到 ALU 的 Y 端，连接后就会出现 Y 输入端的未知状态。

解决方案：在 RAM 输出端与 ALU 的 Y 端之间加一个用作缓冲的寄存器，存放从 RAM 中读出的数据，在给出读地址的时候才刷新寄存器中的数据，就可以解决因 RAM 地址导致的 ALU 的 Y 端的数据出现问题。

2.5 测试与分析

寄存器文件模块测试用例：写入测试如图 2.7 所示；读取测试如图 2.8 和图 2.9 所示；使能信号测试如图 2.10 所示。

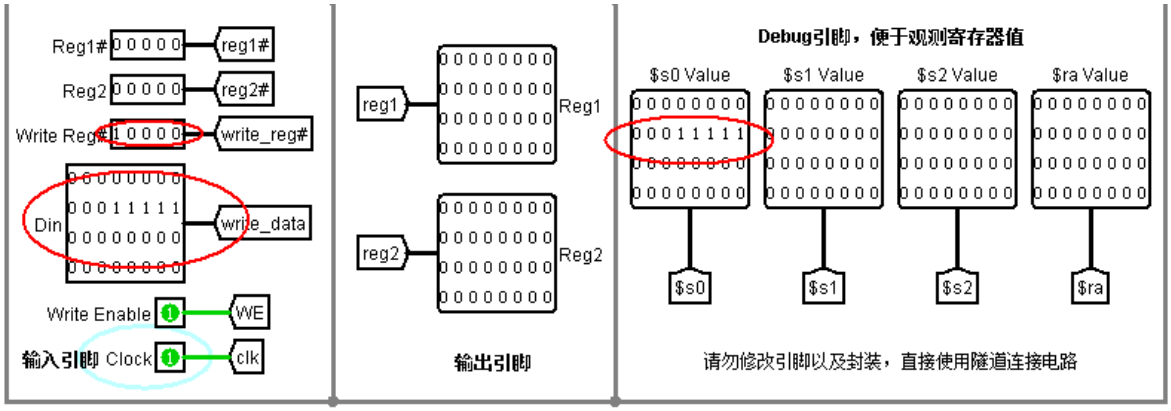


图 2.7 写入测试

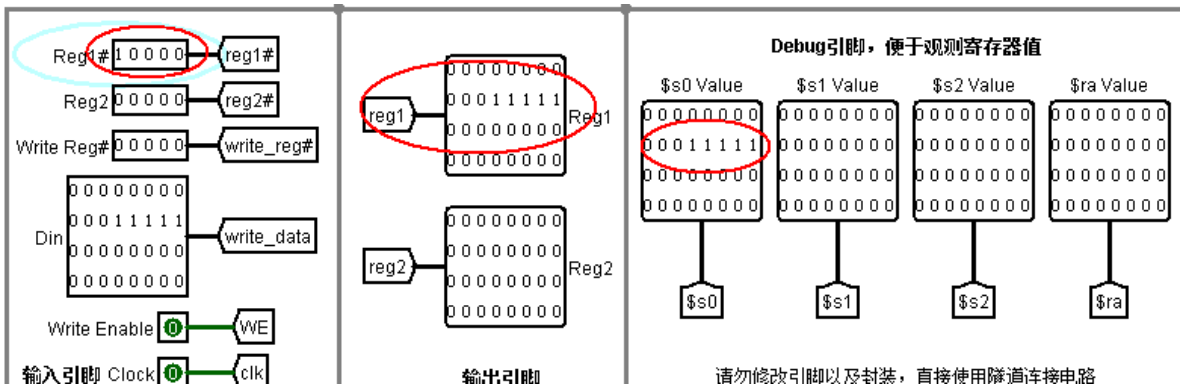


图 2.8 reg1 读取测试

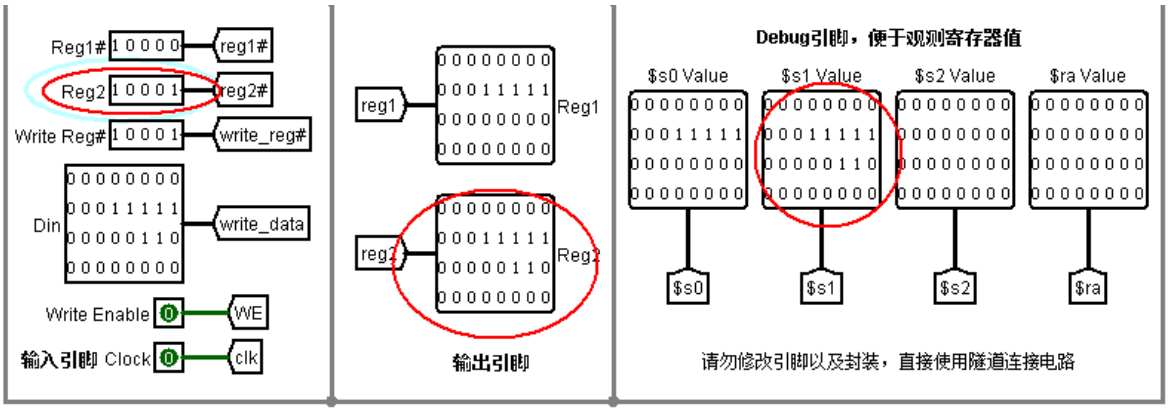


图 2.9 reg2 读取测试

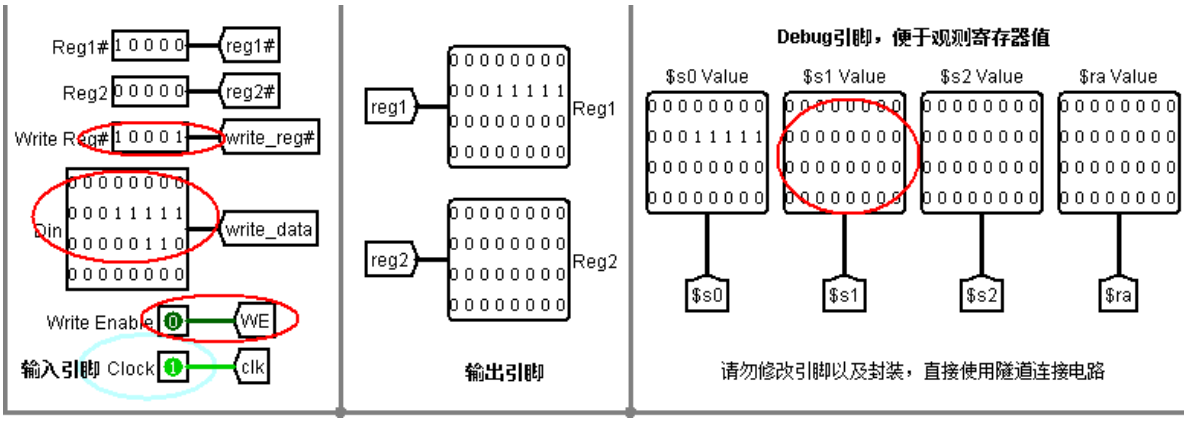


图 2.10 使能信号为 0 时不允许写入

自动运算电路的测试用例，32 个内存单元的数据依次为十进制的 0，1，2，3，……，30，31。模拟运行结果如图 2.11 和图 2.12 所示，计算的中间结果为 0，1，3，6，……，1d1，1f0，最终求和结果位 1F0。

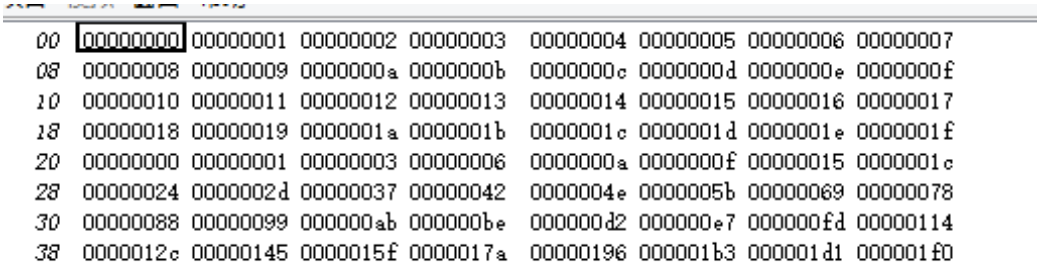


图 2.11 运算中间结果

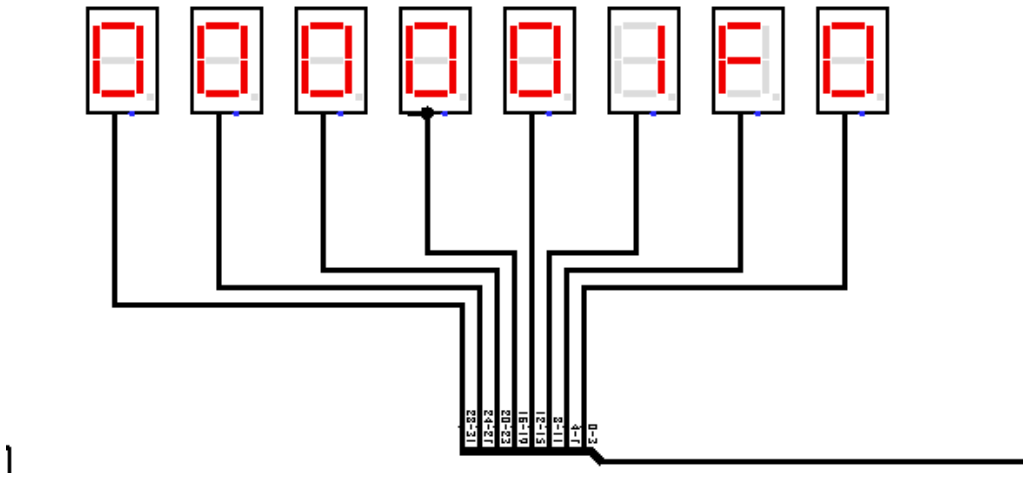


图 2.12 运算最终结果

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），需要完成的指令格式如表 3.1 所示。

3.2 方案设计

3.2.1 指令译码

将指令的 6 位 OP 和 6 位 FUNC 作为输入，24 条指令是否是当前被解释的这一条作为输出，用不同指令的不同 OP 与 FUNC 的对应关系，把每条指令的表达式输入到软件中，自动产生电路。

3.2.2 控制单元

控制单元需要根据不同的指令产生控制信号，比如 ALU 的 op，各个多路选择器的选择信号等

3.2.3 整体结构

CPU 整体应该包括指令存储区 ROM，主存 RAM，寄存器文件，ALU，控制单元，多个多路选择器（跳转时修改 PC，立即数与寄存器值的选择，寄存器号的选择，是否读写内存的选择等），位扩展器等。按照指令执行的顺序构建数据通路，需要时增加必要的多路选择器来增加功能。整体结构如图 3.1 所示：

表 3.1 指令格式

#	指令	格式	备注
1	Add	add $rd, $rs, $rt	
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	指令功能及指令格式 参考 MIPS32 指令集
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 $v0==10 halt(停机指令) else 数码管显示$a0 值

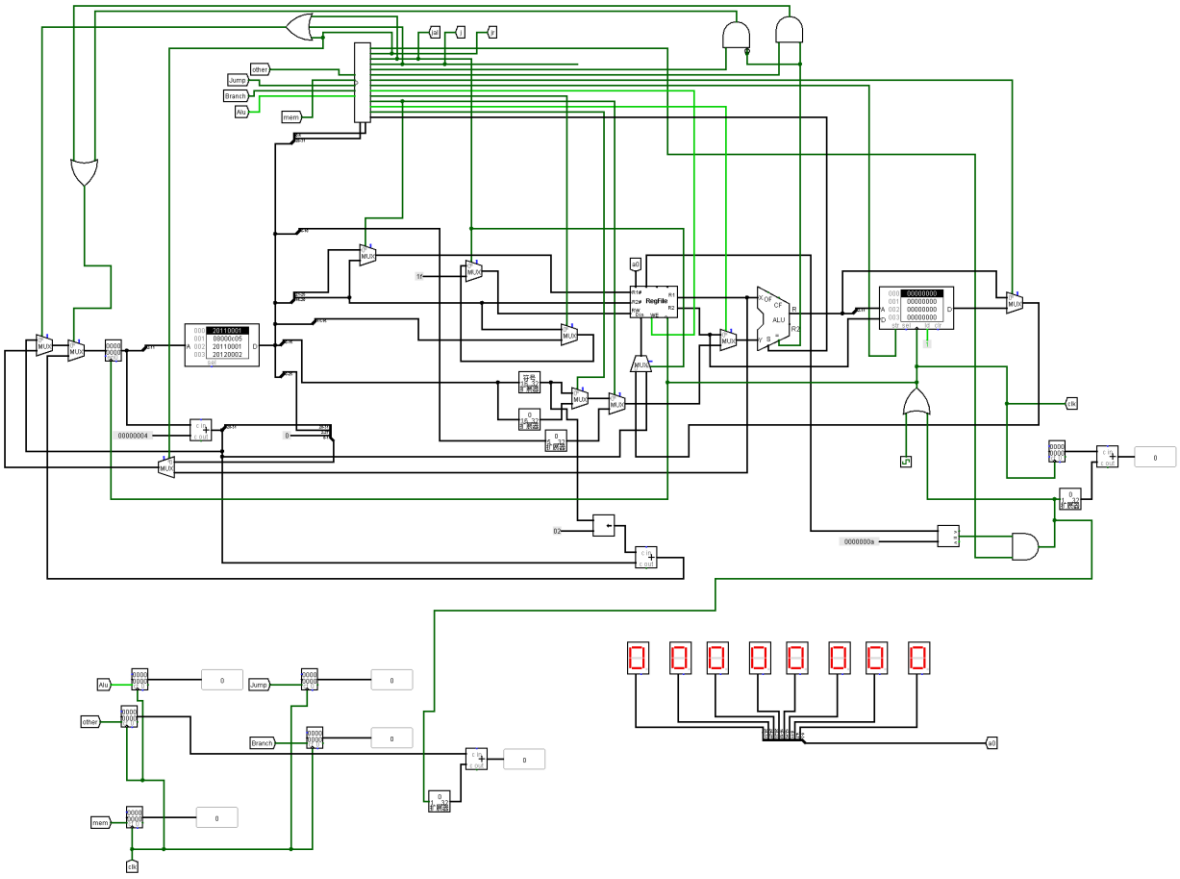


图 3.1 CPU 总体结构

3.3 实验步骤

（1） 指令译码

根据每条指令对应的 op 和 func 位写出不同的表达式，输入到软件中自动构建出指令译码电路，但是一个电路中自动生成最多只能有 12 个输出，所以要把 24 条指令分为两部分生成，然后再把两个电路合并到一个电路里，直接复制粘贴电路到新的电路，并把其中一个的输入端口删除，用隧道与另一个部分的输入端口相连。

（2） 控制单元

控制单元需要根据指令的不同而输出不同的控制信号，包括 ALU 的 op

信号， 无符号扩展，符号扩展，alu 的 Y 值来源，目标寄存器，寄存器写信号，内存写信号，内存数据写入寄存器等等很多。开始的想法是逐条指令产生不同的控制信号，比如先把 add 指令的 op 给出，再把 sub 的 op 指令给出，但后来发现这样做后还要把信号的每一位再分开太过繁琐，所以恰当的做法是每条指令产生很多 1 位的控制信号，即需要为 1 的用线连出来，与其他指令的该信号用或门连出，最后把 4 位 ALU 的 op 信号用分线器连接起来，跳转、系统调用等暂时不能确定具体控制信息的指令直接加输出端口连出，等待后续的使用。

（3） 总体结构

把主要的部件添加到主电路中，然后进行连接。PC（有一个寄存器来保存其值）的 2~11 位连接到 ROM 的地址端，PC 每次加 4 后送回 PC 作为下一条指令地址，此外还有来自指令的立即数经过拼接，来自寄存器文件的输出，立即数扩展移位后与 PC 相加等也可能作为 PC 的值；

指令从 ROM 中取出后分为多个部分，0~5 和 26~11 为 OP 和 FUNC 送到控制单元进行指令译码；0~15 送到位两类扩展器扩展为 32 位，作为 ALU 的 Y 端数据的一种选择；6~10 为移位操作的操作数，经过扩展后也送到 ALU 的 Y 端；21~25,16~20,11~15 都可以当作寄存器的编号，需要经过选择信号的选择后送到寄存器文件的 R1#，R2#，RW 等输入端；寄存器文件的数据输入端要连接 ALU 的运算结果输出端，也要连接 PC，因为有些跳转需要保存 PC 的值；寄存器文件的两个输出端，其一连接 ALU 的 X 端，其二连接 ALU 的 Y 端或 RAM 的写入数据端（写内存的指令需要）；

ALU 的运算结果输出连接到 RAM 的地址端或寄存器文件的数据写入端；RAM 的输出端连接到寄存器文件的数据写入端；

停机指令需要判断寄存器文件$v0 的值是否为 10，考虑到之前的寄存器文件有四个输出端口是没有用到的，所以可以把其中一个输出端口改为输出$v0 的值，用来与 10 比较，所以这次的寄存器文件与之前的文件是略有不同的；

统计指令的条数。因为是单周期的 CPU，只需要统计时钟的周期数就可

以，需要注意的是最后一条停机指令在时钟被屏蔽后才起了作用，需要最后再加 1 才是真正的指令条数；各类别指令用的分类方法是分为 Alu, Branch, memery, Jump 和 Other 五种，具体的操作是在控制单元部件中增加几个输出端口，来表示是某一条指令是什么类型的，相同类型指令的输出用或门组合;在 main 电路中把增加的控制信号连接到对应计数器的计数端，Other 型指令同样存在未统计到停机指令的问题，需要最后再加 1；

3.4 故障与调试

3.4.1 进行排序测时写入的 RAM 单元有问题

故障现象： 排序测试时，排序的结果是正确的，但是写入的 RAM 单元不对，每四个单元才会写入一个数据，如图 3.2 所示

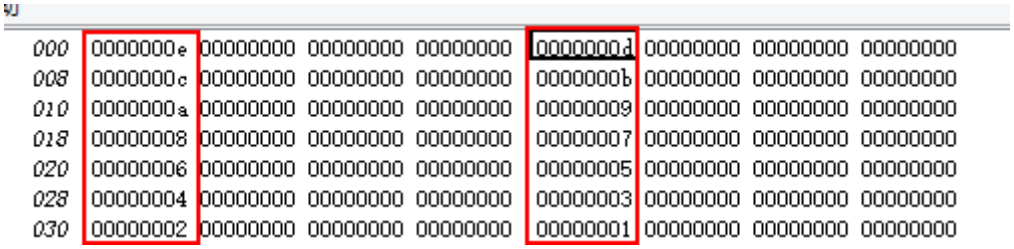


图 3.2 排序故障

原因分析： 写入的 RAM 单元不对应当是 RAM 的地址线连接出现问题，检查地址线的连接，果然是把 ALU 运算结果截取的位弄错了，2~11 连成了 0~9。

解决方案： 把错误连接的 0~9 位改为正确的 2~11 位。

3.4.2 统计指令条数不对

故障现象：统计指令的总条数时，统计的结果是 1545，比 benchmark 的 1546 少一条，如图 3.3 所示。

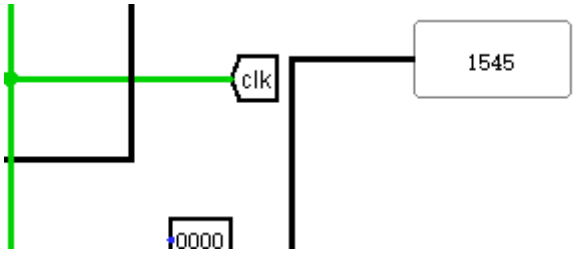


图 3.3 指令条数故障

原因分析：经过分析计数的线路特点，发现是最后的停机指令没有统计进去。

解决方案：试过把计数器的上升沿触发改为下降沿触发，但是还是不正确，所以采用了更加直接的方法，在最后用一个加法器把计数器的值加 1，Other 类型指令的问题同样采用此方法解决。

3.4.3 统计分类指令条数出现错误

故障现象：统计的各类指令条数与 benchmark 的运行结果有很大差别，多数指令的数目是偏少的。

原因分析：从控制单元连出各类指令的信号后，直接把信号连到了计数器的时钟端，因为有些连续的指令可能类型相同，就不会引起时钟沿的变化，导致计数偏少，只有没有相邻的情况下才能计数。

解决方案：计数器的时钟端与系统的时钟端相连，用统一信号，把从控制单元输出的信号连到对应计数器的计数端，当技术信号为 1 时，计数器可以在时钟的触发下计数，这样就可以正确计数了。

3.5 测试与分析

系统对 benchmark 的测试结果：

RAM 内容如图 3.4 所示；

运行过程中一个状态如图 3.5 所示；

最后的数码管显示如图 3.6 所示；

总指令条数如图 3.7；

各类指令条数如图 3.8 所示。

```
000  0000000e 0000000d 0000000c 0000000b
004  0000000a 00000009 00000008 00000007
008  00000006 00000005 00000004 00000003
00c  00000002 00000001 00000000 ffffffff
010  00000000 00000000 00000000 00000000
014  00000000 00000000 00000000 00000000
```

图 3.4 排序后 RAM 中的内容

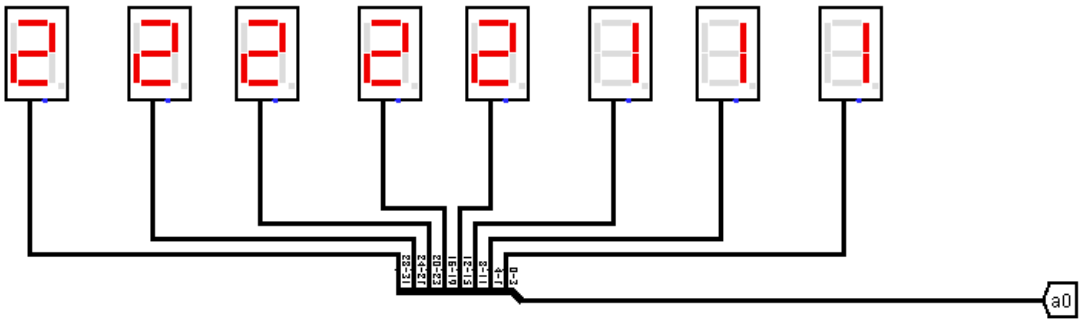


图 3.5 运行过程中的一个状态

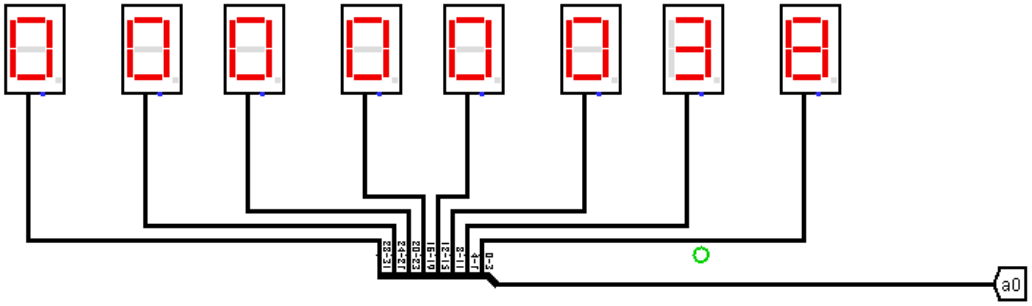


图 3.6 最后的数码管显示

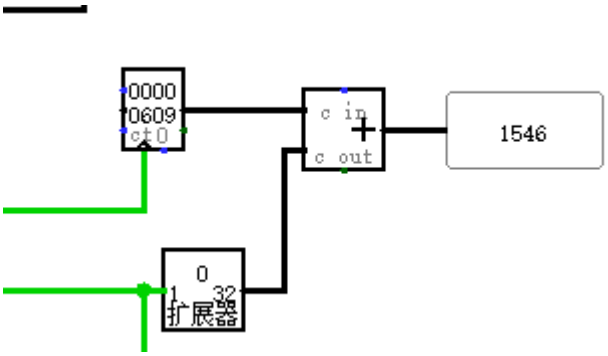


图 3.7 总指令条数

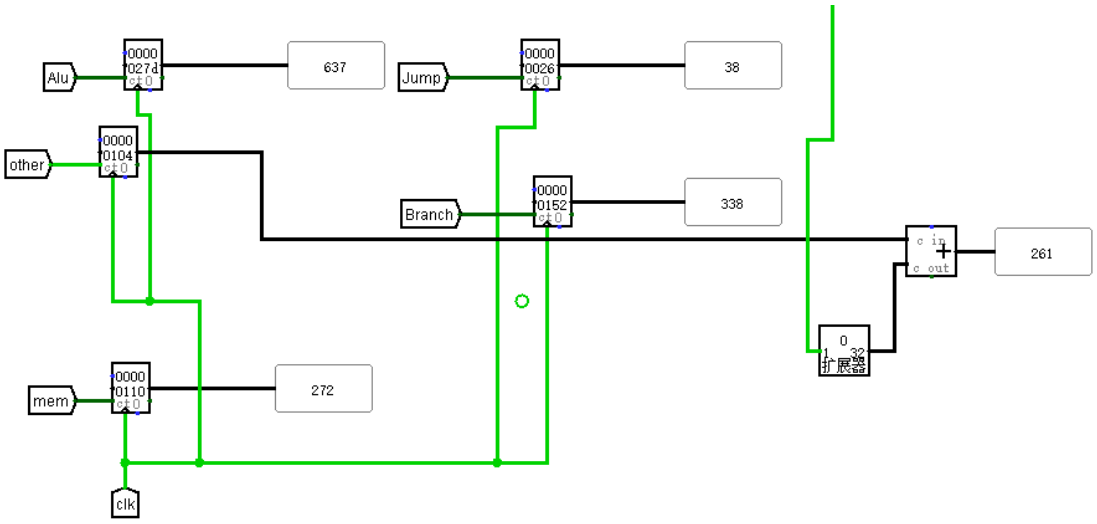


图 3.8 统计各类指令条数

4 总结与心得

4.1 实验总结

本次实验主要完成了如下几点工作：

1） 完成 32 位快速加法器的设计，完成运算器 ALU 的封装；

完成包含 32 个 32 位寄存器的 MIPS 寄存器组的封装，完成由时钟驱动的自动运算电路；

完成指令译码电路，完成控制信号生成电路，完成支持 24 条 MIPS 指令的 CPU。

2） 功能总结：32 位快速加法器可以计算两个 32 位数据（包括有符号和无符号）的和，并给出是否溢出的判断（区分有符号溢出和无符号溢出）；ALU 可以进行逻辑左移、算数右移、逻辑右移、有符号乘、无符号除、加、减、按位与、按位或、按位异或、按位或非、无符号比较和有符号比较 13 种运算并给出是否溢出、相等的检测；

寄存器组文件可以在写使能和时钟的控制下把输入端数据写入到指定号的寄存器中，也可以读出给定号的寄存器中的内容，自动运算电路可自动完成 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；

MIPS 处理器支持运行 add，addi，addiu，……，syscall 等 24 条指令的执行，并统计所运行的总指令条数与各类指令条数。

3） 不足之处：寄存器组文件中写入数据没有必要使用解复用器，可以直接连接到寄存器的数据输入端，只由写使能信号控制是否写入；

CPU 设计的模块化程度不足，多路选择器的使用方法不合适，在实验中使用了过多的二路选择器，把功能相同的选择信号组合起来使用多路选择器代替多个二路选择器会更条理，更清晰；取出指令后对指令进行分割，分割的顺序可以调整一下，减少不必要的线路；CPU 整体的布局不好看，占据空间太多，隧道使用比较少，连线过多导致整体结构看上去有点凌乱，

虽然实现了完整的功能但调试的时候会比较费时间。

4.2 实验心得

1） 通过这次实验，自己收获了很多，首先进位是学会了使用 logisim 这种工具，其次把课堂学的内容运用到实践中，加深了对课堂内容的理解。

2） 设计完整的 CPU 才让自己深入了解了计算机运行程序的过程，以前也只是学习汇编语言和数字逻辑了解了一些较为底层的东西，并没有把两者之间关联起来，但通过这次实验真正学习到了机器指令是如何在计算机上执行的，每条指令的执行都是一个独特的数据通路，由统一的时钟信号驱动执行；程序的指令和数据都保存在存储元件中，需要用到时候通过不同的寻址方式访问数据。

3） 调试错误时增强了自己的耐心，看到满屏的线和元件虽然头都大了，但还是要一点一点去找错误的原因，修改再测试；连线的过程也一样，稍不注意可能就把线连错了，对全局的把握能力也有提升。

4） 感受到 CPU 设计真的是一件麻烦的事情，每条指令都有自己的数据通路，实现各自的功能，设计的时候就要用很多元件把不同指令数据通路的不同部分分开，而且每增加一条指令就有可能要把过去连过的线路大改重连，元件的摆放也需要考虑好，在后续增加指令时如果元件摆放位置不合适会多画很多的线路，如果使用太多隧道也会造成问题，就是看不出元件之间的连接关系，虽然看上去整洁但也不完美，就是设计过程中需要协调的东西有很多。

5） 对实验的建议，感觉实验的难度不是很大，只要是多学多问就可以做出来，但是会比较耗时间，而且三次实验耗时是依次增加的，仅靠课表安排的时间是远远不够的，课下需要做很多工作，希望可以适当延长一些实验时间吧，总体感觉实验的收获还是很大的。

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: 许荣仁

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**