实验三报告

231220088 陈翔宇

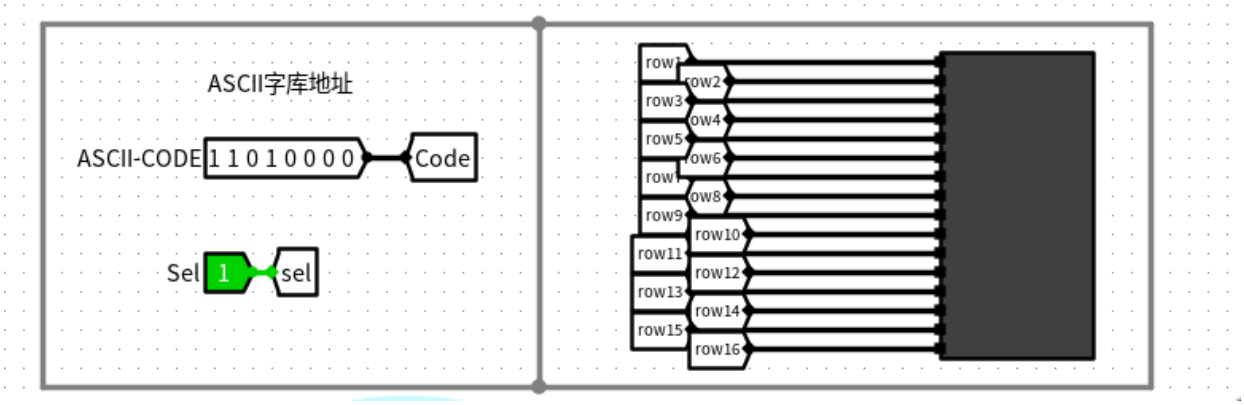
实验内容

一、 *只读存储器实验*

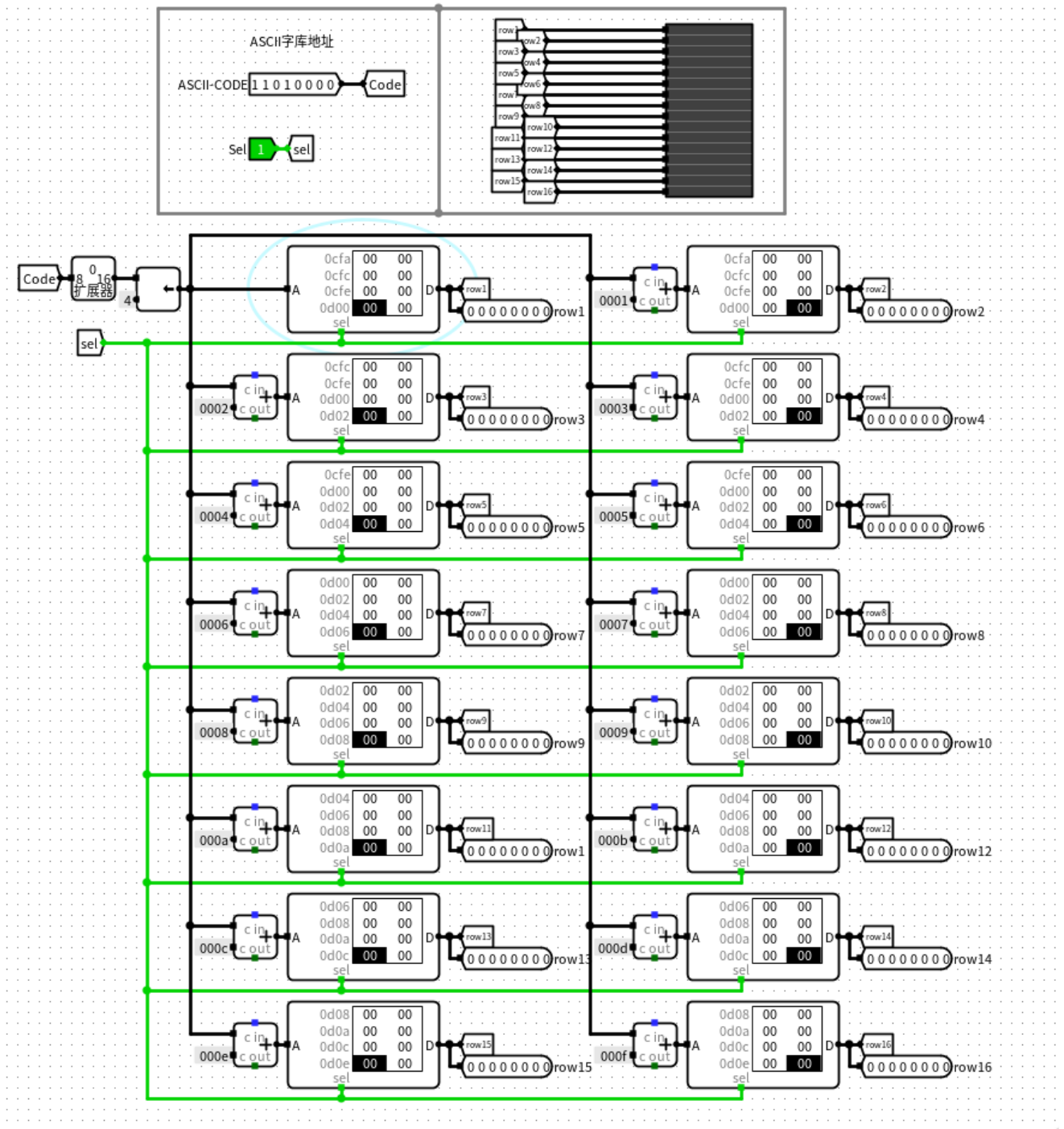
整体方案设计

首行是 Logisim 数据文件的标识，不存储到存储器中；第二行表示字符空格的字形点阵，第三行表示字符“！”的字形点阵，第四行表示字符“”“的字形点阵，以此类推；载入存储器中时从第二行开始。每个字符的字形点阵都是 16 个字节，表示 16 行 8 列矩阵，如果某 1 位为 1，则表示对应矩阵点是可显示的。

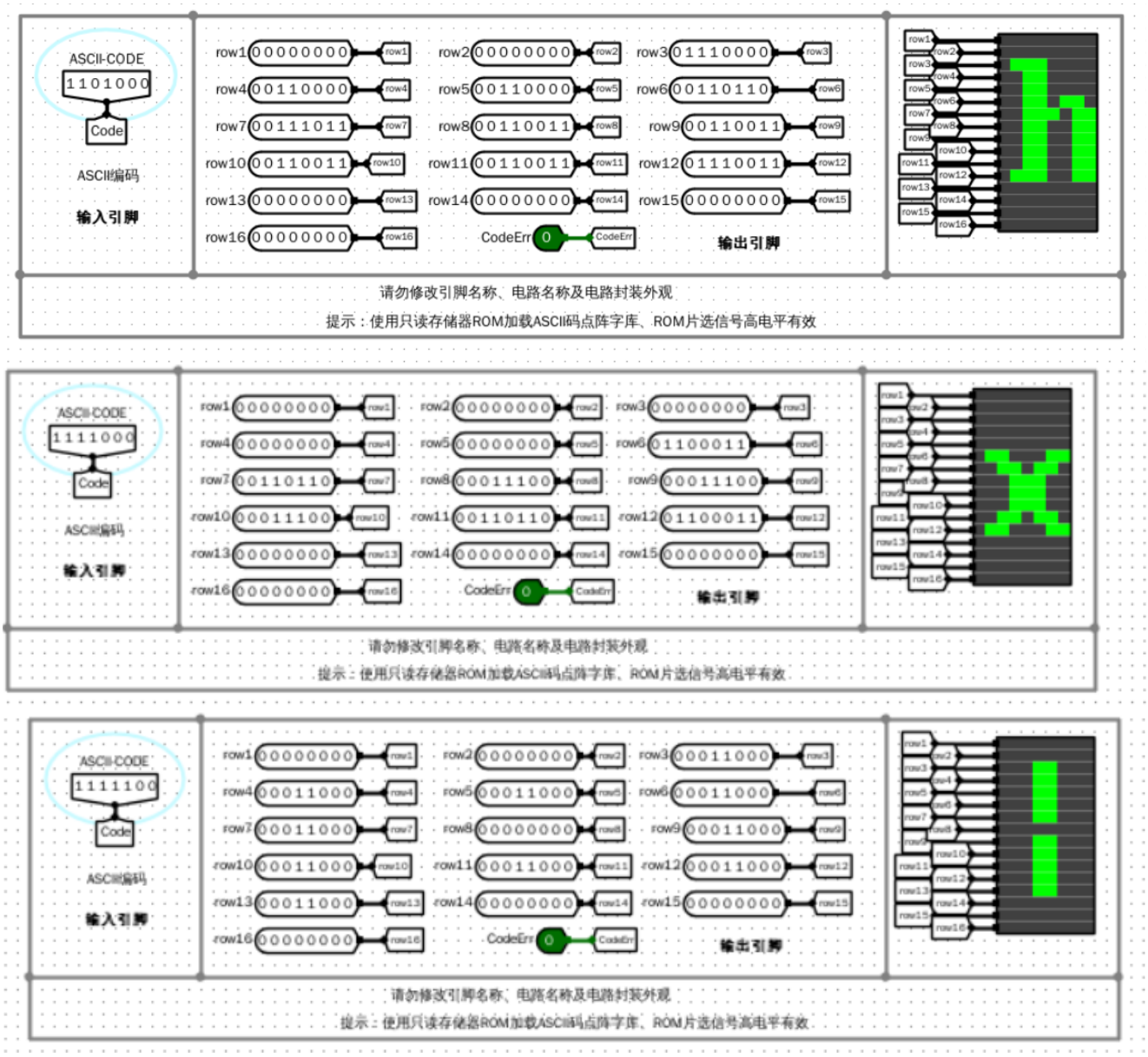
1. 输入输出引脚



电路图 和 原理图


ASCII字库地址
ASCII-CODE
1 1 0 1 0 0 0 0
Code
Sel
1
sel
Code
8
16
扩展器
4
sel
A
D
sel
c in
c out
0001
0002
0003
0004
0005
0006
0007
0008
0009
000a
000b
000c
000d
000e
000f
row1
row2
row3
row4
row5
row6
row7
row8
row9
row10
row11
row12
row13
row14
row15
row16

仿真测试



错误现象及分析

在完成实验过程中没有遇到任何错误。

二、数据存储器实验

基本原理

表 5.1 MemOp 控制信号含义

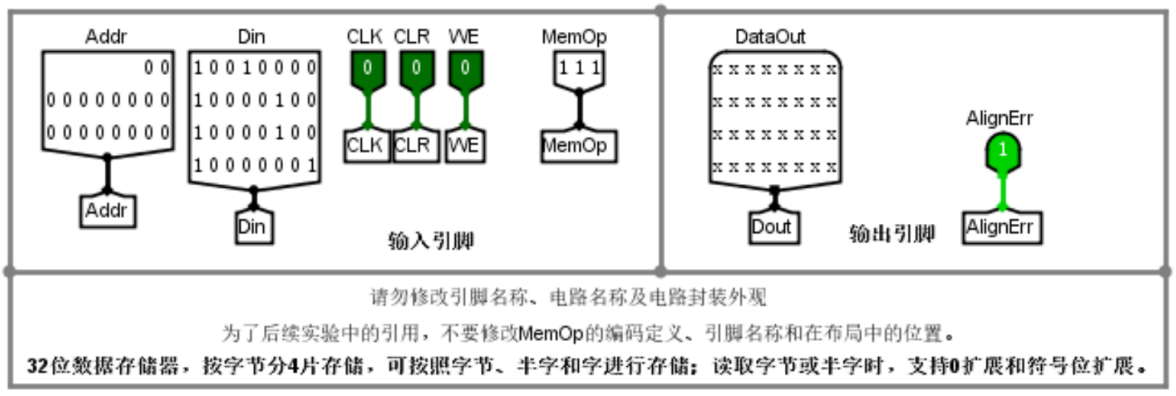
MemOp	指令	含　义
000	lw,sw	按字存取，4 字节
001	lbu	按字节读取，1 个字节，0 扩展到 4 字节
010	lhu	按半字读取，2 字节，0 扩展到 4 字节
101	lb,sb	按字节存取，1 个字节，在读取时，按符号位扩展到 4 字节
110	lh,sh	按半字存取，2 个字节，在读取时，按符号位扩展到 4 字节

表 5.2 存储器控制信号、地址低 2 位和片选信号、地址对齐的对应关系

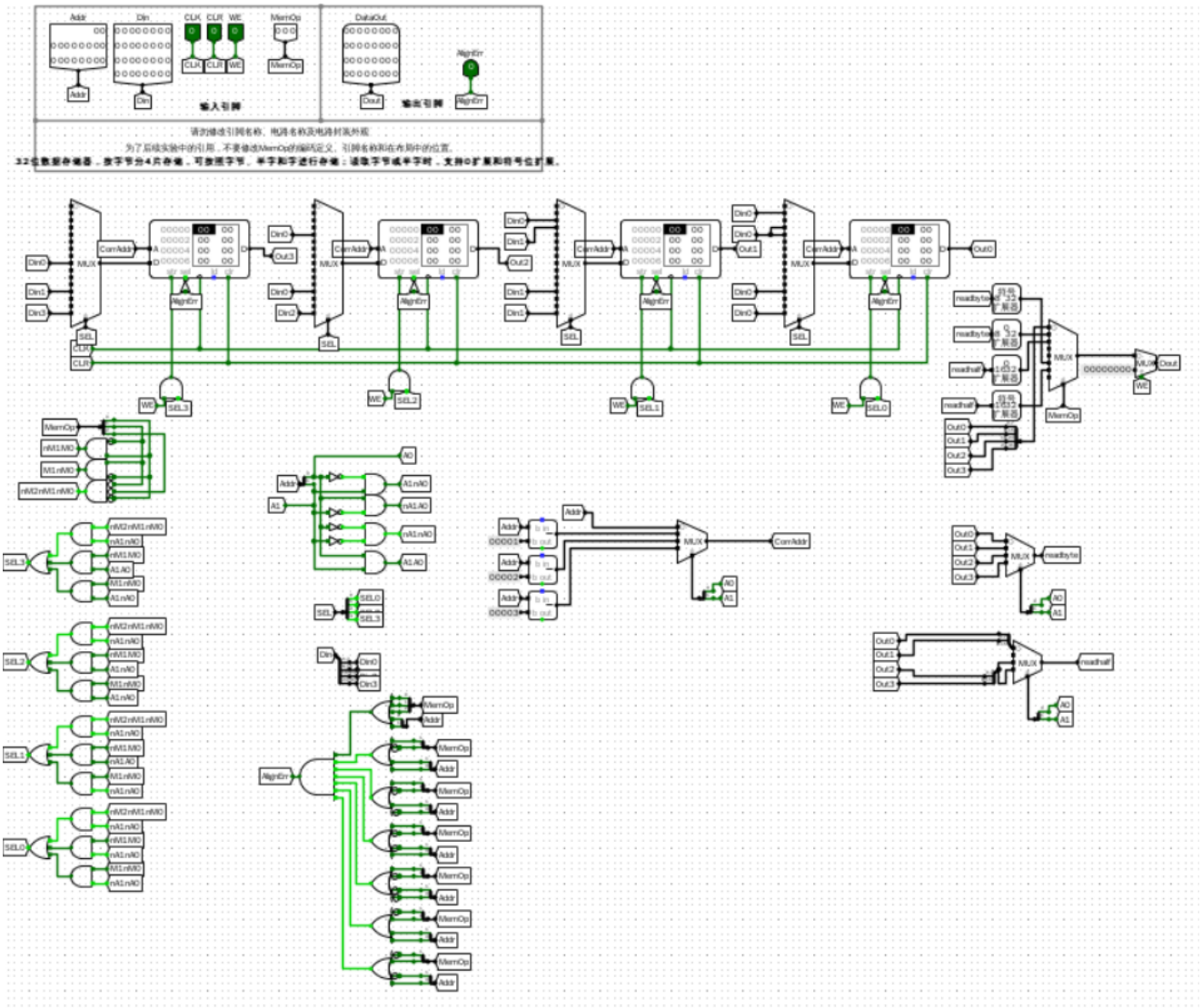
MemOp[2][1][0]	Addr[1][0]	SEL3	SEL2	SEL1	SEL0	AlignErr
000	00	1	1	1	1	0
*00	01	0	0	0	0	1
*00	10	0	0	0	0	1
*00	11	0	0	0	0	1
*01	00	0	0	0	1	0
*01	01	0	0	1	0	0
*01	10	0	1	0	0	0
*01	11	1	0	0	0	0
*10	00	0	0	1	1	0
*10	01	0	0	0	0	1
*10	10	1	1	0	0	0
*10	11	0	0	0	0	1
其它	**	0	0	0	0	1

整体方案设计

1. 输入输出引脚



电路图


Addr
Din
CLK
CLR
WE
MemOp
DataOut
AlignErr
输入引脚
输出引脚
请勿修改引脚名称，电路名称及电路封装外观
为了后续实验中的引用，不要修改MemOp的编码定义，引脚名称和在布局中的位置。
32位数据存储器，按字节分4片存储，可按照字节、半字和字进行存储；读取字节或半字时，支持0扩展和符号位扩展。
MUX
ComAddr
Dout
SEL
SEL3
SEL2
SEL1
SEL0
readbyte
readbyteu
readhalf
readhalfu

仿真测试

—— 预期输出 ——

Cnt	Addr	WE	CLR	MemOp	Din	Dout
00	00000	1	0	0	00000001	00000000
01	00000	0	0	0	005deece	00000001
02	00010	1	0	5	b61488df	00000000
03	00011	1	0	5	f4111591	00000000
04	00012	1	0	5	023eaf12	00000000
05	00013	1	0	5	b578fa6a	00000000
06	00010	0	0	0	d38a8b1c	6a1291df
07	00100	1	0	6	f5d50649	00000000
08	00102	1	0	6	202e3c08	00000000
09	00100	0	0	0	812fba12	3c080649
0a	00100	0	0	1	6755ebab	00000049
0b	00101	0	0	1	f6e7cab1	00000006
0c	00102	0	0	1	e6bc21b9	00000008
0d	00103	0	0	1	a6c5abc6	0000003c
0e	00100	0	0	2	58228a26	00000649
0f	00102	0	0	2	9dfc1362	00003c08
10	01000	1	0	0	ac5b2754	00000000
11	01000	0	0	5	2f43454c	00000054
12	01001	0	0	5	18c3dc9c	00000027
13	01002	0	0	5	1abbd85c	0000005b
14	01003	0	0	5	cb1b519a	fffffffac
15	01000	0	0	3	0f672c6a	xxxxxxxx
16	01000	0	0	4	e37362d1	xxxxxxxx
17	01000	0	0	7	39eaeb41	xxxxxxxx
18	01001	0	0	6	95c7a81f	xxxxxxxx
19	01003	0	0	6	9b8ab32e	xxxxxxxx
1a	01001	0	0	0	64bb0d7c	xxxxxxxx
1b	01002	0	0	0	72f86d97	xxxxxxxx

—— 实际输出 ——

展示原始输出

Cnt	Addr	WE	CLR	MemOp	Din	Dout
00	00000	1	0	0	00000001	00000000
01	00000	0	0	0	005deece	00000001
02	00010	1	0	5	b61488df	00000000
03	00011	1	0	5	f4111591	00000000
04	00012	1	0	5	023eaf12	00000000
05	00013	1	0	5	b578fa6a	00000000
06	00010	0	0	0	d38a8b1c	6a1291df
07	00100	1	0	6	f5d50649	00000000
08	00102	1	0	6	202e3c08	00000000
09	00100	0	0	0	812fba12	3c080649
0a	00100	0	0	1	6755ebab	00000049
0b	00101	0	0	1	f6e7cab1	00000006
0c	00102	0	0	1	e6bc21b9	00000008
0d	00103	0	0	1	a6c5abc6	0000003c
0e	00100	0	0	2	58228a26	00000649
0f	00102	0	0	2	9dfc1362	00003c08
10	01000	1	0	0	ac5b2754	00000000
11	01000	0	0	5	2f43454c	00000054
12	01001	0	0	5	18c3dc9c	00000027
13	01002	0	0	5	1abbd85c	0000005b
14	01003	0	0	5	cb1b519a	fffffffac
15	01000	0	0	3	0f672c6a	xxxxxxxx
16	01000	0	0	4	e37362d1	xxxxxxxx
17	01000	0	0	7	39eaeb41	xxxxxxxx
18	01001	0	0	6	95c7a81f	xxxxxxxx
19	01003	0	0	6	9b8ab32e	xxxxxxxx
1a	01001	0	0	0	64bb0d7c	xxxxxxxx
1b	01002	0	0	0	72f86d97	xxxxxxxx

错误现象及分析

在完成实验过程中没有遇到任何错误。

三、取指令部件实验

基本原理

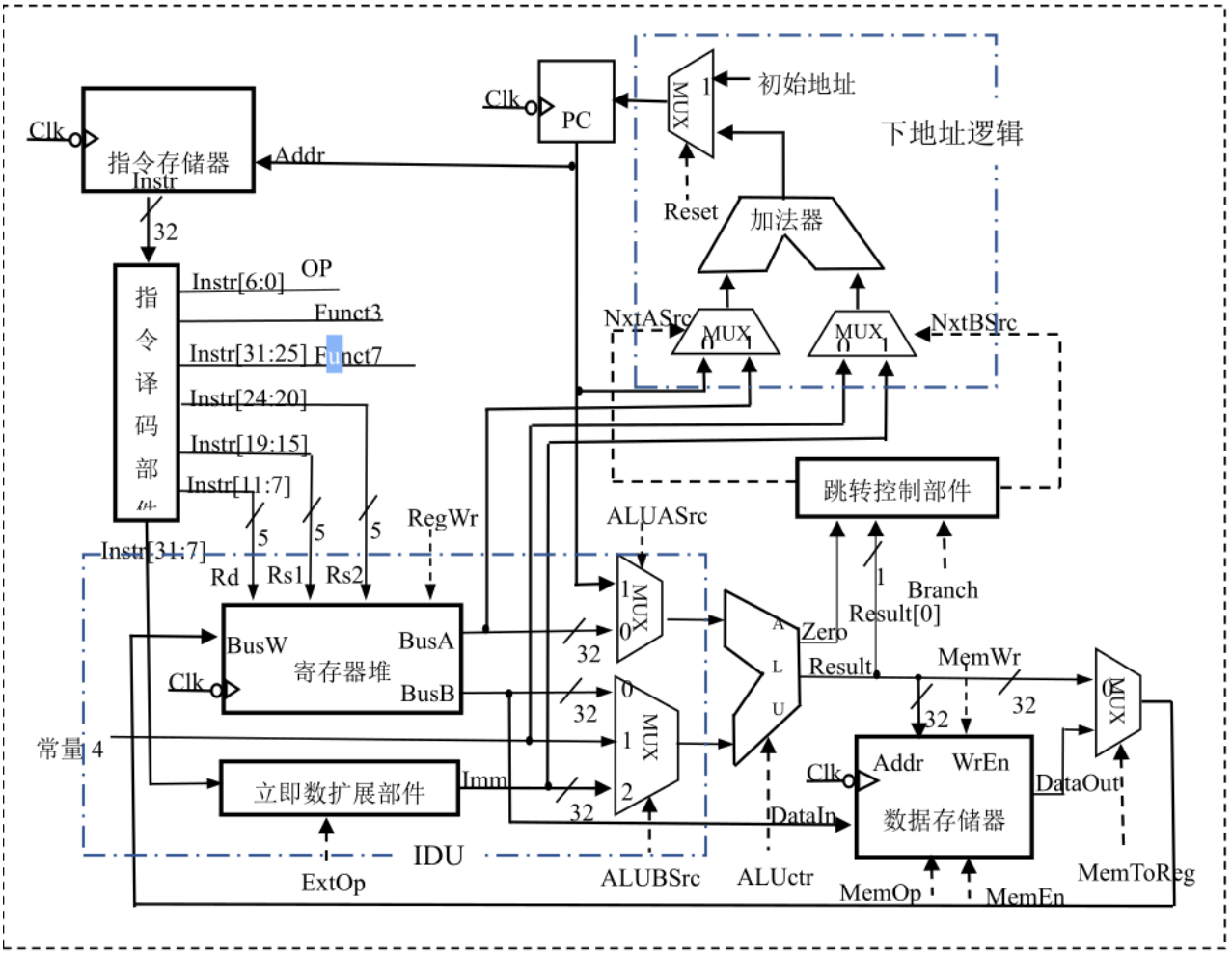
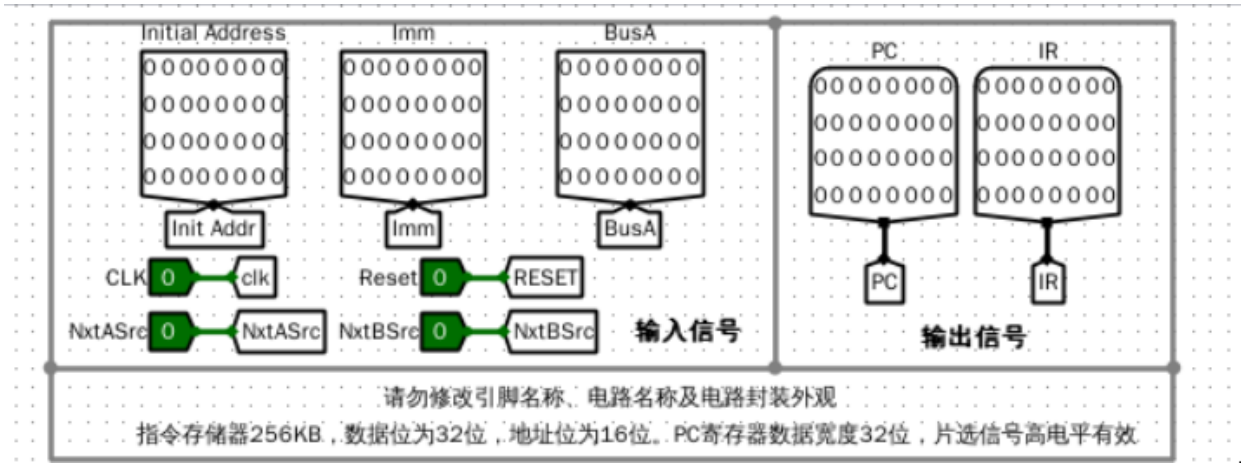


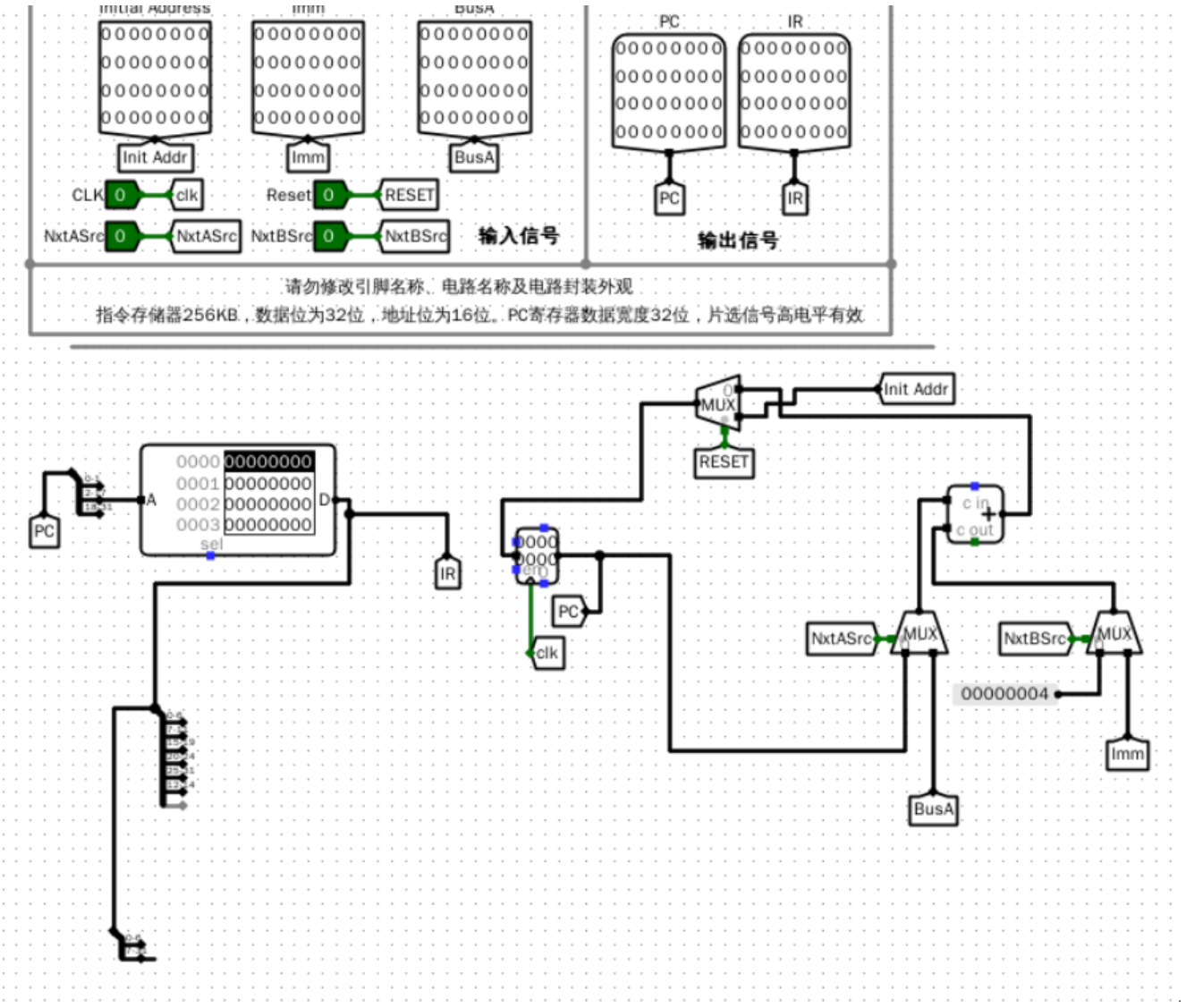
图 5.7　单周期 CPU 数据通路原理图

整体方案设计

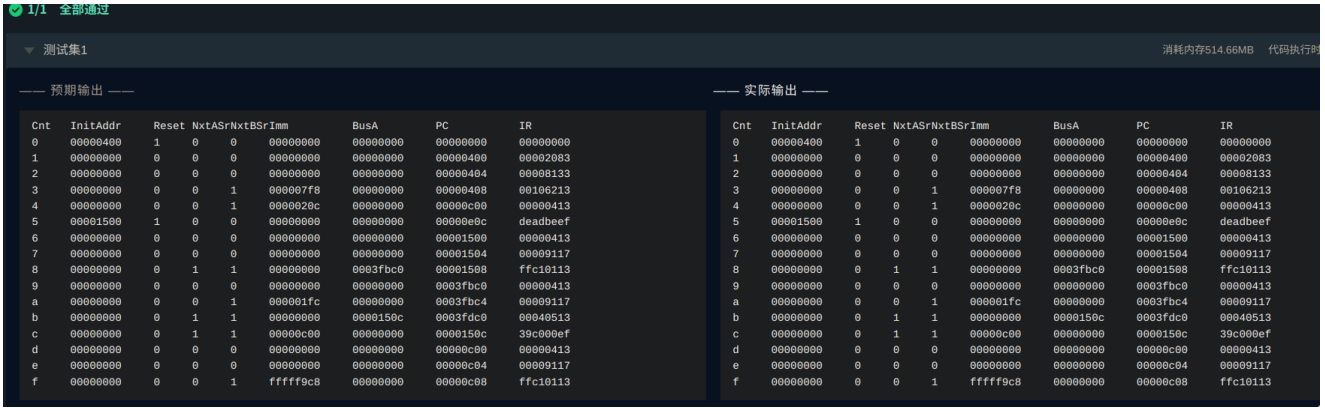
1. 输入输出引脚



电路图



仿真测试



1/1 全部通过

测试集1

消耗内存514.66MB 代码执行时

—— 预期输出 ——

Cnt	InitAddr	Reset	NxtASr	NxtBSr	Imm	BusA	PC	IR
0	00000400	1	0	0	00000000	00000000	00000000	00000000
1	00000000	0	0	0	00000000	00000000	00000400	00002083
2	00000000	0	0	0	00000000	00000000	00000404	00008133
3	00000000	0	0	1	000007f8	00000000	00000408	00106213
4	00000000	0	0	1	0000020c	00000000	00000c00	00000413
5	00001500	1	0	0	00000000	00000000	00000e0c	deadbeef
6	00000000	0	0	0	00000000	00000000	00001500	00000413
7	00000000	0	0	0	00000000	00000000	00001504	00009117
8	00000000	0	1	1	00000000	0003fbc0	00001508	ffc10113
9	00000000	0	0	0	00000000	00000000	0003fbc0	00000413
a	00000000	0	0	1	000001fc	00000000	0003fbc4	00009117
b	00000000	0	1	1	00000000	0000150c	0003fdc0	00040513
c	00000000	0	1	1	00000c00	00000000	0000150c	39c000ef
d	00000000	0	0	0	00000000	00000000	00000c00	00000413
e	00000000	0	0	0	00000000	00000000	00000c04	00009117
f	00000000	0	0	1	fffff9c8	00000000	00000c08	ffc10113

—— 实际输出 ——

Cnt	InitAddr	Reset	NxtASr	NxtBSr	Imm	BusA	PC	IR
0	00000400	1	0	0	00000000	00000000	00000000	00000000
1	00000000	0	0	0	00000000	00000000	00000400	00002083
2	00000000	0	0	0	00000000	00000000	00000404	00008133
3	00000000	0	0	1	000007f8	00000000	00000408	00106213
4	00000000	0	0	1	0000020c	00000000	00000c00	00000413
5	00001500	1	0	0	00000000	00000000	00000e0c	deadbeef
6	00000000	0	0	0	00000000	00000000	00001500	00000413
7	00000000	0	0	0	00000000	00000000	00001504	00009117
8	00000000	0	1	1	00000000	0003fbc0	00001508	ffc10113
9	00000000	0	0	0	00000000	00000000	0003fbc0	00000413
a	00000000	0	0	1	000001fc	00000000	0003fbc4	00009117
b	00000000	0	1	1	00000000	0000150c	0003fdc0	00040513
c	00000000	0	1	1	00000c00	00000000	0000150c	39c000ef
d	00000000	0	0	0	00000000	00000000	00000c00	00000413
e	00000000	0	0	0	00000000	00000000	00000c04	00009117
f	00000000	0	0	1	fffff9c8	00000000	00000c08	ffc10113

错误现象及分析

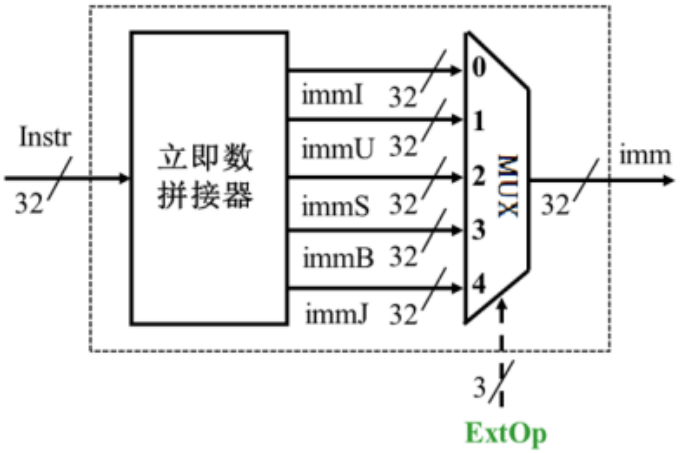
在完成实验过程中没有遇到任何错误。

四、取操作数部件 IDU 实验

基本原理

```
immI = {20{Instr[31]}, Instr[31:20]};
immU = {Instr[31:12], 12'b0};
immS = {20{Instr[31]}, Instr[31:25], Instr[11:7]};
immB = {19{Instr[31]},Instr[31], Instr[7], Instr[30:25], Instr[11:8], 1'b0};
immJ = {11{Instr[31]}, Instr[31], Instr[19:12], Instr[20], Instr[30:21], 1'b0};
```

其设计示意图如图 5.9 所示，通过控制信号 ExtOp 来选择不同立即数编码类型以及在扩展器中进行的扩展操作。

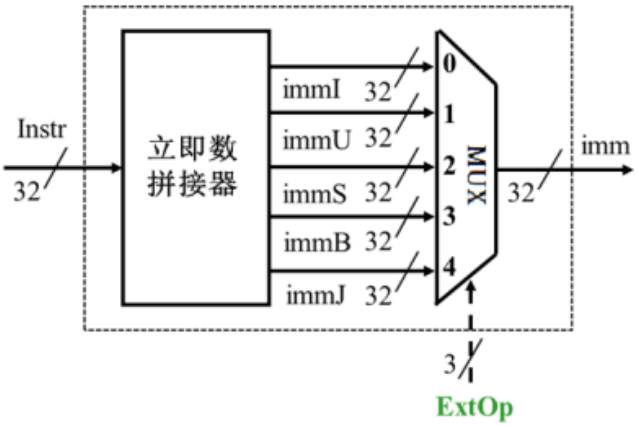


整体方案设计

1. 输入输出引脚

```
immI = {20{Instr[31]}, Instr[31:20]};
immU = {Instr[31:12], 12'b0};
immS = {20{Instr[31]}, Instr[31:25], Instr[11:7]};
immB = {19{Instr[31]},Instr[31], Instr[7], Instr[30:25], Instr[11:8], 1'b0};
immJ = {11{Instr[31]}, Instr[31], Instr[19:12], Instr[20], Instr[30:21], 1'b0};
```

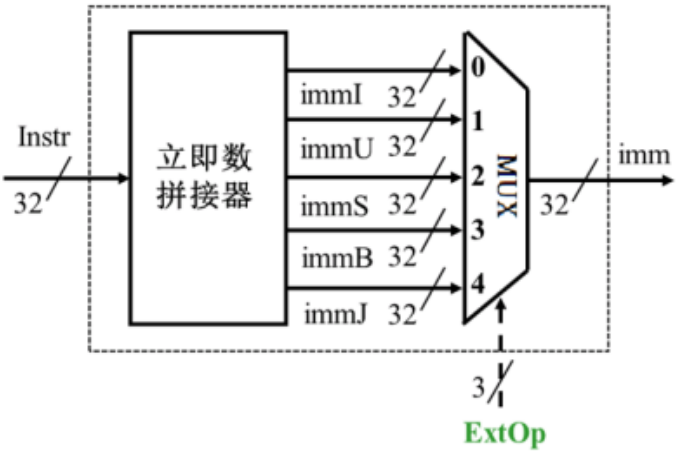
其设计示意图如图 5.9 所示，通过控制信号 ExtOp 来选择不同立即数编码类型以及在扩展器中进行的扩展操作。



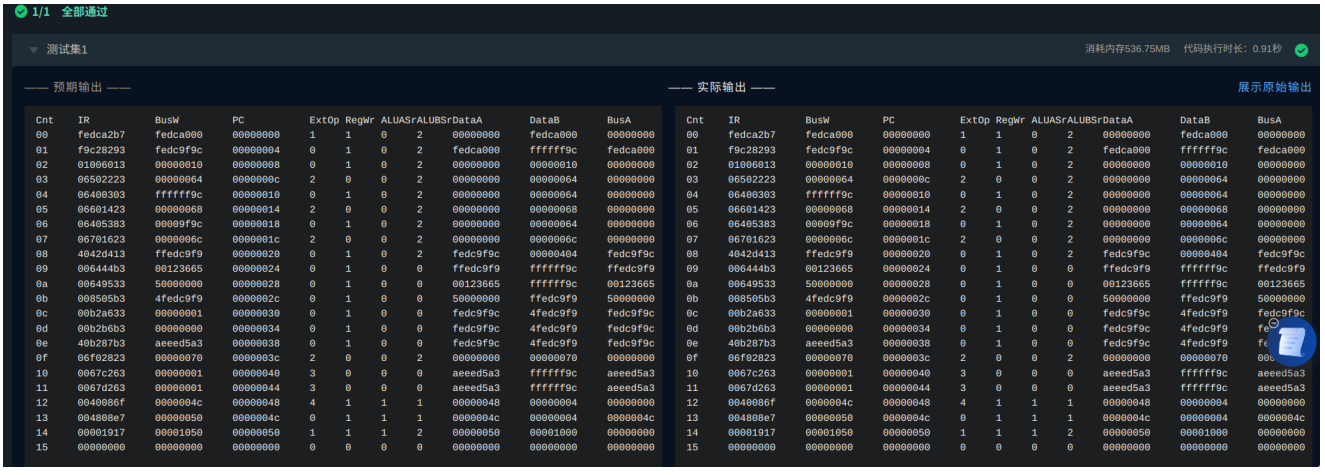
电路图

immI = {20{Instr[31]}, Instr[31:20]};
immU = {Instr[31:12], 12'b0};
immS = {20{Instr[31]}, Instr[31:25], Instr[11:7]};
immB = {19{Instr[31]},Instr[31], Instr[7], Instr[30:25], Instr[11:8], 1'b0};
immJ = {11{Instr[31]}, Instr[31], Instr[19:12], Instr[20], Instr[30:21], 1'b0};

其设计示意图如图 5.9 所示，通过控制信号 ExtOp 来选择不同立即数编码类型以及在扩展器中进行的扩展操作。



仿真测试



1/1 全部通过

测试集1　　消耗内存536.75MB　代码执行时长：0.91秒

预期输出

Cnt	IR	BusW	PC	ExtOp	RegWr	ALUASrc	ALUBSrc	DataA	DataB	BusA
00	fedca2b7	fedca000	00000000	1	1	0	2	00000000	fedca000	00000000
01	f9c28293	fedc9f9c	00000004	0	1	0	2	fedca000	ffffff9c	fedca000
02	01006013	00000010	00000008	0	1	0	2	00000000	00000010	00000000
03	06502223	00000064	0000000c	2	0	0	2	00000000	00000064	00000000
04	06400303	ffffff9c	00000010	0	1	0	2	00000000	00000064	00000000
05	06601423	00000068	00000014	2	0	0	2	00000000	00000068	00000000
06	06405383	00009f9c	00000018	0	1	0	2	00000000	00000064	00000000
07	06701623	0000006c	0000001c	2	0	0	2	00000000	0000006c	00000000
08	4042d413	ffedc9f9	00000020	0	1	0	2	fedc9f9c	00000404	fedc9f9c
09	006444b3	00123665	00000024	0	1	0	0	ffedc9f9	ffffff9c	ffedc9f9
0a	00649533	50000000	00000028	0	1	0	0	00123665	ffffff9c	00123665
0b	008505b3	4fedc9f9	0000002c	0	1	0	0	50000000	ffedc9f9	50000000
0c	00b2a633	00000001	00000030	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c
0d	00b2b6b3	00000000	00000034	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c
0e	40b287b3	aeeed5a3	00000038	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c
0f	06f02823	00000070	0000003c	2	0	0	2	00000000	00000070	00000000
10	0067c263	00000001	00000040	3	0	0	0	aeeed5a3	ffffff9c	aeeed5a3
11	0067d263	00000001	00000044	3	0	0	0	aeeed5a3	ffffff9c	aeeed5a3
12	0040086f	0000004c	00000048	4	1	1	1	00000048	00000004	00000000
13	004808e7	00000050	0000004c	0	1	1	1	0000004c	00000004	0000004c
14	00001917	00001050	00000050	1	1	1	2	00000050	00001000	00000000
15	00000000	00000000	00000000	0	0	0	0	00000000	00000000	00000000

实际输出　　展示原始输出

Cnt	IR	BusW	PC	ExtOp	RegWr	ALUASrc	ALUBSrc	DataA	DataB	BusA
00	fedca2b7	fedca000	00000000	1	1	0	2	00000000	fedca000	00000000
01	f9c28293	fedc9f9c	00000004	0	1	0	2	fedca000	ffffff9c	fedca000
02	01006013	00000010	00000008	0	1	0	2	00000000	00000010	00000000
03	06502223	00000064	0000000c	2	0	0	2	00000000	00000064	00000000
04	06400303	ffffff9c	00000010	0	1	0	2	00000000	00000064	00000000
05	06601423	00000068	00000014	2	0	0	2	00000000	00000068	00000000
06	06405383	00009f9c	00000018	0	1	0	2	00000000	00000064	00000000
07	06701623	0000006c	0000001c	2	0	0	2	00000000	0000006c	00000000
08	4042d413	ffedc9f9	00000020	0	1	0	2	fedc9f9c	00000404	fedc9f9c
09	006444b3	00123665	00000024	0	1	0	0	ffedc9f9	ffffff9c	ffedc9f9
0a	00649533	50000000	00000028	0	1	0	0	00123665	ffffff9c	00123665
0b	008505b3	4fedc9f9	0000002c	0	1	0	0	50000000	ffedc9f9	50000000
0c	00b2a633	00000001	00000030	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c
0d	00b2b6b3	00000000	00000034	0	1	0	0	fedc9f9c	4fedc9f9	fe
0e	40b287b3	aeeed5a3	00000038	0	1	0	0	fedc9f9c	4fedc9f9	fe
0f	06f02823	00000070	0000003c	2	0	0	2	00000000	00000070	00
10	0067c263	00000001	00000040	3	0	0	0	aeeed5a3	ffffff9c	aeeed5a3
11	0067d263	00000001	00000044	3	0	0	0	aeeed5a3	ffffff9c	aeeed5a3
12	0040086f	0000004c	00000048	4	1	1	1	00000048	00000004	00000000
13	004808e7	00000050	0000004c	0	1	1	1	0000004c	00000004	0000004c
14	00001917	00001050	00000050	1	1	1	2	00000050	00001000	00000000
15	00000000	00000000	00000000	0	0	0	0	00000000	00000000	00000000

五、数据通路实验

基本原理

数据通路是具体完成数据存取、运算的部件。单周期 CPU 的数据通道是指获取到指令之后，根据指令内容，读取操作数，进行操作，得到结果并写回的过程。不同类型的指令，数据传输过程并不一致。大致可分为取指令 IFU、取操作数 IDU、执行指令 EX、访问存储器 M 和写回寄存器堆 WB 等阶段。支持 RV32I 中不同类型指令的单周期数据通路电路原理图如图 5.7 所示，主要部件在前述实验中已基本完成，如在 lab4.5 中完成了 32 位 ALU 的设计，在 lab5.2 中完成数据存储器的设计，在 lab5.3 中完成了取指令部件的设计，在 lab5.4 中完成了取操作数部件 IDU 的设计。在本次实验中需要先完成跳转控制部件的设计。

1） 设计跳转控制器子电路

跳转控制器根据控制信号 Branch 和 ALU 输出的 Zero 及 Result[0]信号来决定 NxtASrc 和 NxtBSrc，其中控制信号 Branch 的定义来自于跳转指令，编码定义如表 5.4 所示。提示：为了后续实验中的子电路直接引用，不要修改编码定义。

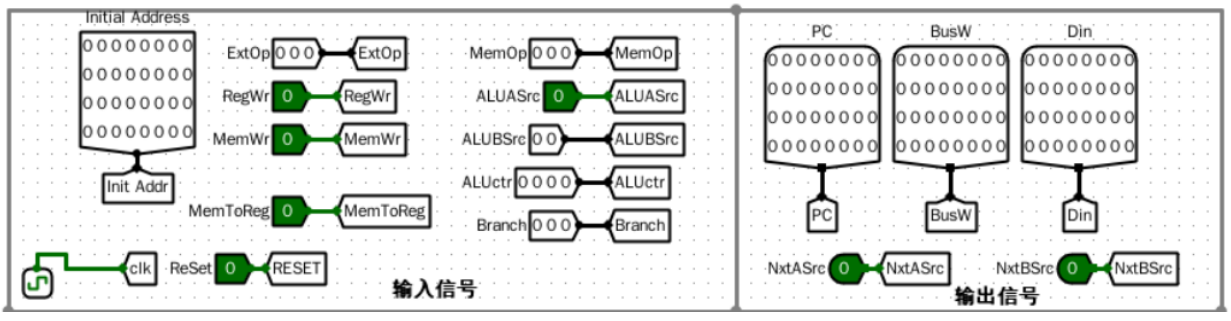
表 5.4 Branch 控制信号含义

Branch	NxtASrc	NxtBSrc	指令跳转类型
000	0	0	非跳转指令
001	0	1	jal：无条件跳转 PC 目标
010	1	1	jalr：无条件跳转寄存器目标
100	0	Zero	beq：条件分支，等于
101	0	! Zero	bne：条件分支，不等于
110	0	Result[0]	blt,bltu：条件分支，小于
111	0	Zero | (! Result[0])	bge,bgeu：条件分支，大于等于

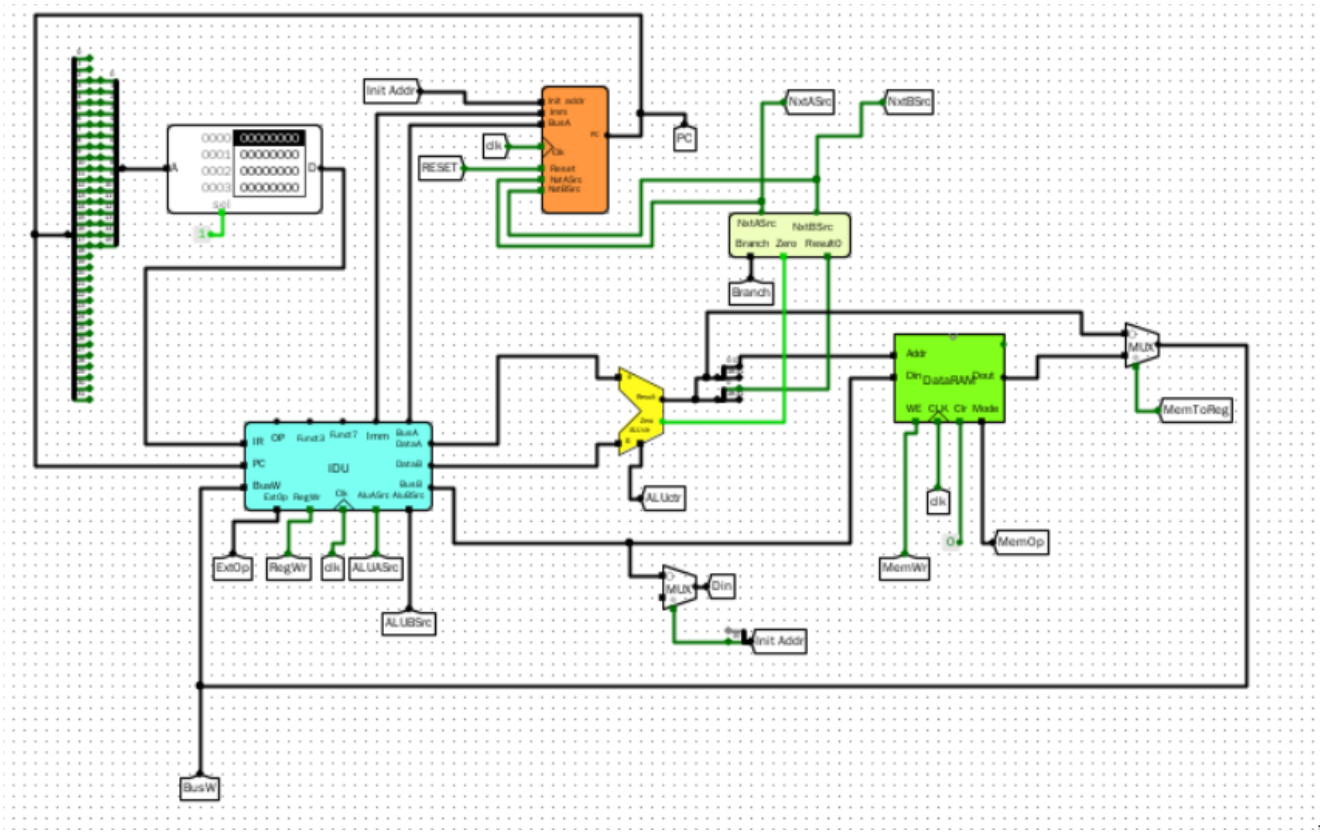
在 Logisim 中添加一个名为“Branch”的子电路，双击该子电路名称，在右侧工作区中构建相应电路。引脚参考如图 5.15 所示，根据表 5.4 所示，列出 NxtASrc 和 NxtBSrc 两个信号的逻辑表达式，在工作区中

整体方案设计

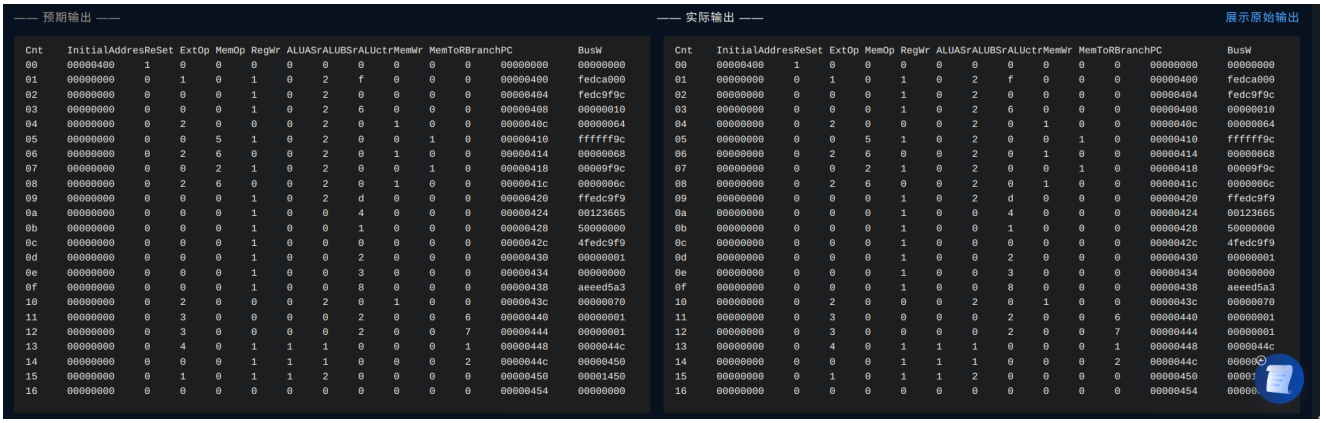
1. 输入输出引脚



电路图



仿真测试



预期输出

Cnt	InitialAddresReSet	ExtOp	MemOp	RegWr	ALUASrALUBSrALUctrMemWr	MemToR	BranchPC	BusW
00	00000400	1	0	0	0 0 0 0 0 0	0	00000000	00000000
01	00000000	0	1	0	1 0 2 f 0 0	0	00000400	fedca000
02	00000000	0	0	0	1 0 2 0 0 0	0	00000404	fedc9f9c
03	00000000	0	0	0	1 0 2 6 0 0	0	00000408	00000010
04	00000000	0	2	0	0 0 2 0 1 0	0	0000040c	00000064
05	00000000	0	0	5	1 0 2 0 0 1	0	00000410	ffffff9c
06	00000000	0	2	6	0 0 2 0 1 0	0	00000414	00000068
07	00000000	0	0	2	1 0 2 0 0 1	0	00000418	00009f9c
08	00000000	0	2	6	0 0 2 0 1 0	0	0000041c	0000006c
09	00000000	0	0	0	1 0 2 d 0 0	0	00000420	ffedc9f9
0a	00000000	0	0	0	1 0 0 4 0 0	0	00000424	00123665
0b	00000000	0	0	0	1 0 0 1 0 0	0	00000428	50000000
0c	00000000	0	0	0	1 0 0 0 0 0	0	0000042c	4fedc9f9
0d	00000000	0	0	0	1 0 0 2 0 0	0	00000430	00000001
0e	00000000	0	0	0	1 0 0 3 0 0	0	00000434	00000000
0f	00000000	0	0	0	1 0 0 8 0 0	0	00000438	aeeed5a3
10	00000000	0	2	0	0 0 2 0 1 0	0	0000043c	00000070
11	00000000	0	3	0	0 0 0 2 0 0	6	00000440	00000001
12	00000000	0	3	0	0 0 0 2 0 0	7	00000444	00000001
13	00000000	0	4	0	1 1 1 0 0 0	1	00000448	0000044c
14	00000000	0	0	0	1 1 1 0 0 0	2	0000044c	00000450
15	00000000	0	1	0	1 1 2 0 0 0	0	00000450	00001450
16	00000000	0	0	0	0 0 0 0 0 0	0	00000454	00000000

实际输出

Cnt	InitialAddresReSet	ExtOp	MemOp	RegWr	ALUASrALUBSrALUctrMemWr	MemToR	BranchPC	BusW
00	00000400	1	0	0	0 0 0 0 0 0	0	00000000	00000000
01	00000000	0	1	0	1 0 2 f 0 0	0	00000400	fedca000
02	00000000	0	0	0	1 0 2 0 0 0	0	00000404	fedc9f9c
03	00000000	0	0	0	1 0 2 6 0 0	0	00000408	00000010
04	00000000	0	2	0	0 0 2 0 1 0	0	0000040c	00000064
05	00000000	0	0	5	1 0 2 0 0 1	0	00000410	ffffff9c
06	00000000	0	2	6	0 0 2 0 1 0	0	00000414	00000068
07	00000000	0	0	2	1 0 2 0 0 1	0	00000418	00009f9c
08	00000000	0	2	6	0 0 2 0 1 0	0	0000041c	0000006c
09	00000000	0	0	0	1 0 2 d 0 0	0	00000420	ffedc9f9
0a	00000000	0	0	0	1 0 0 4 0 0	0	00000424	00123665
0b	00000000	0	0	0	1 0 0 1 0 0	0	00000428	50000000
0c	00000000	0	0	0	1 0 0 0 0 0	0	0000042c	4fedc9f9
0d	00000000	0	0	0	1 0 0 2 0 0	0	00000430	00000001
0e	00000000	0	0	0	1 0 0 3 0 0	0	00000434	00000000
0f	00000000	0	0	0	1 0 0 8 0 0	0	00000438	aeeed5a3
10	00000000	0	2	0	0 0 2 0 1 0	0	0000043c	00000070
11	00000000	0	3	0	0 0 0 2 0 0	6	00000440	00000001
12	00000000	0	3	0	0 0 0 2 0 0	7	00000444	00000001
13	00000000	0	4	0	1 1 1 0 0 0	1	00000448	0000044c
14	00000000	0	0	0	1 1 1 0 0 0	2	0000044c	00000450
15	00000000	0	1	0	1 1 2 0 0 0	0	00000450	00001450
16	00000000	0	0	0	0 0 0 0 0 0	0	00000454	00000000

思考题

1. 如何利用 ROM 实验实现滚动显示的功能，在 3 个 LED 点阵矩阵中，左右滚动显示 5 个 ASCII 字符，如“NJUCS”

1. 将 ASCII 字符的字模数据存储在 ROM 中，每个字符占用相应的存储空间。
2. 通过控制逻辑电路，按照一定的时序和频率，从 RO M 中读取字模数据，并将其输出到 LED 点阵矩阵中，实现滚动显示效果。
3. 实现步骤包括：读取一个字符的字模数据，将其输出到点阵矩阵上，等待一段时间，
4. 然后将点阵矩阵的显示位置左移一个像素，继续读取下一个字符的字模数据，
5. 重复上述步骤。

2. 分析说明如果寄存器堆写入数据时是下降沿触发有效，而 PC 寄存器和数据存储器写入时是上升沿触发有效，则对程序执行结果有什么影响?

```
时钟信号的上升沿和下降沿触发的时刻不同，会导致不同的数据写入时机。
如果指令的执行过程中存在依赖于上升沿触发的状态，而在该状态更新之前就发生了下降沿触发的寄存器堆写入，就可能导致程序执行结果与预期不符
```

3. 在 CPU 启动执行后，如何实现在当前程序结束后，CPU 不再继续执行指令?

```
一种常见的方式是，在程序的最后添加一个特殊的指令或者标记位置，当 CPU 执行到该指令或者检测到该标记时，就停止继续执行指令，进入一个停止状态。可以通过控制信号或者特定的机器码来触发停止操作，将 CPU 的执行状态置为非运行状态，从而实现停止执行指令的目的。
```