实验 5：存储器及数据通路设计

一、实验目的

1. 学习存储器的读写方法，掌握指令存储器和数据存储器的设计方法。

2. 学习处理器读取指令的过程，掌握 RV32I 下取指令部件设计方法

3. 分析 RV32I 不同运算指令的执行过程，掌握指令译码、取操作数、运算、访存等执行不同阶段的实现方法。

4. 分析单周期 CPU 的设计要求，掌握单周期数据通路的设计方法。

二、实验环境

Logisim 2.16

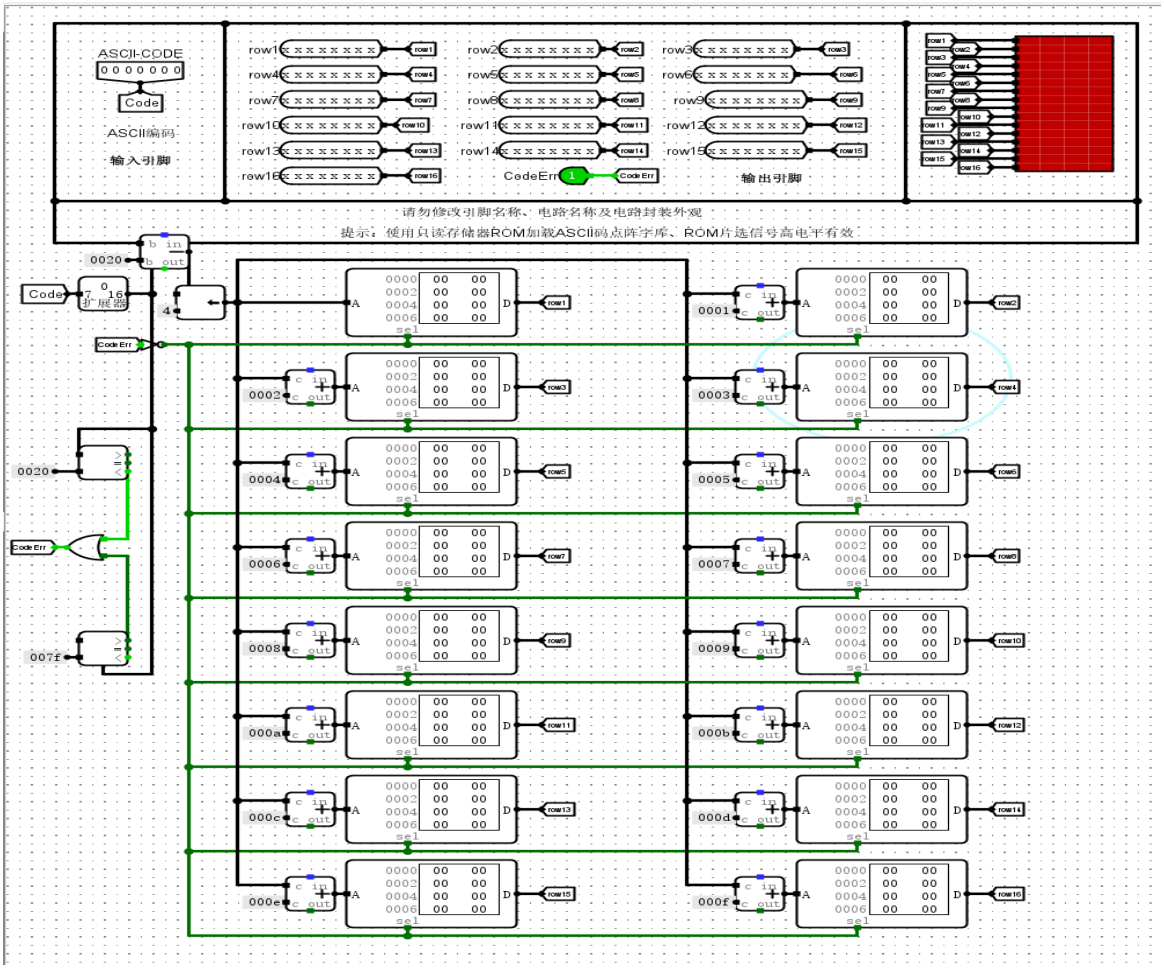
三、实验内容

1. 只读存储器实验

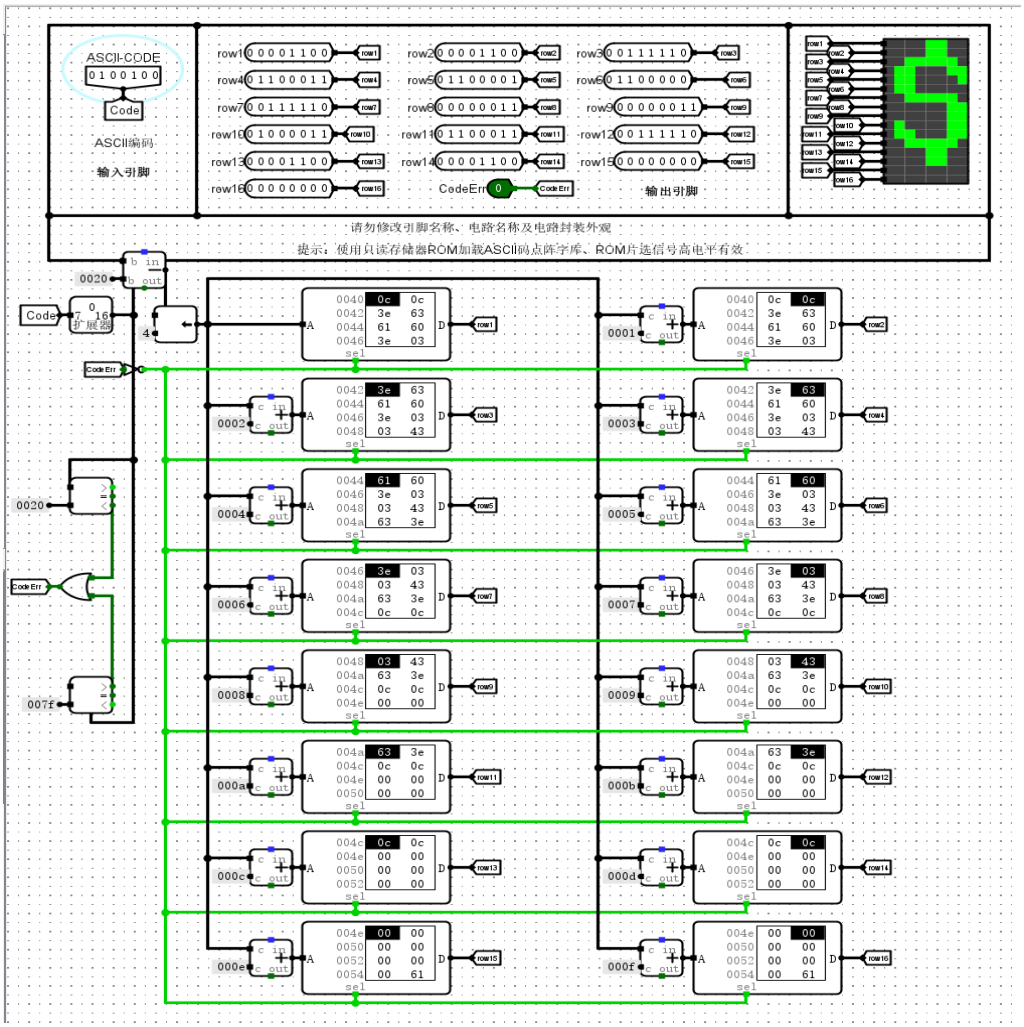
整体模块设计：

从字符空格 SP（ASCII 码值 0x20）开始，到删除字符 DEL（ASCII 码值 0x7f）结束共 96 个字符，每个字符使用 8 列 16 行的点阵表示字形。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。要先把 ASCII 码值减去十六进制 0x20，然后根据每个字符的点阵数据是 16 个字节，把得到的差值左移 4 位在低位补 4 个 0，则得到该字符点阵在字库中的起始位置，字符点阵数据为连续存放的 16 个字节。为了能够同时输出 16 个字节的点阵数据，则需要复 制 16 个加载相同点阵字库的只读存储器，读取点阵数据时每个只读存储器的地址数据按点阵行的次序加 1 递增，则可以同时读取第 1 行、第 2 行、…、第 16 行的点阵数据，然后再同步输出到 LED 点阵组件的输入 端中。

原理图（电路图）：

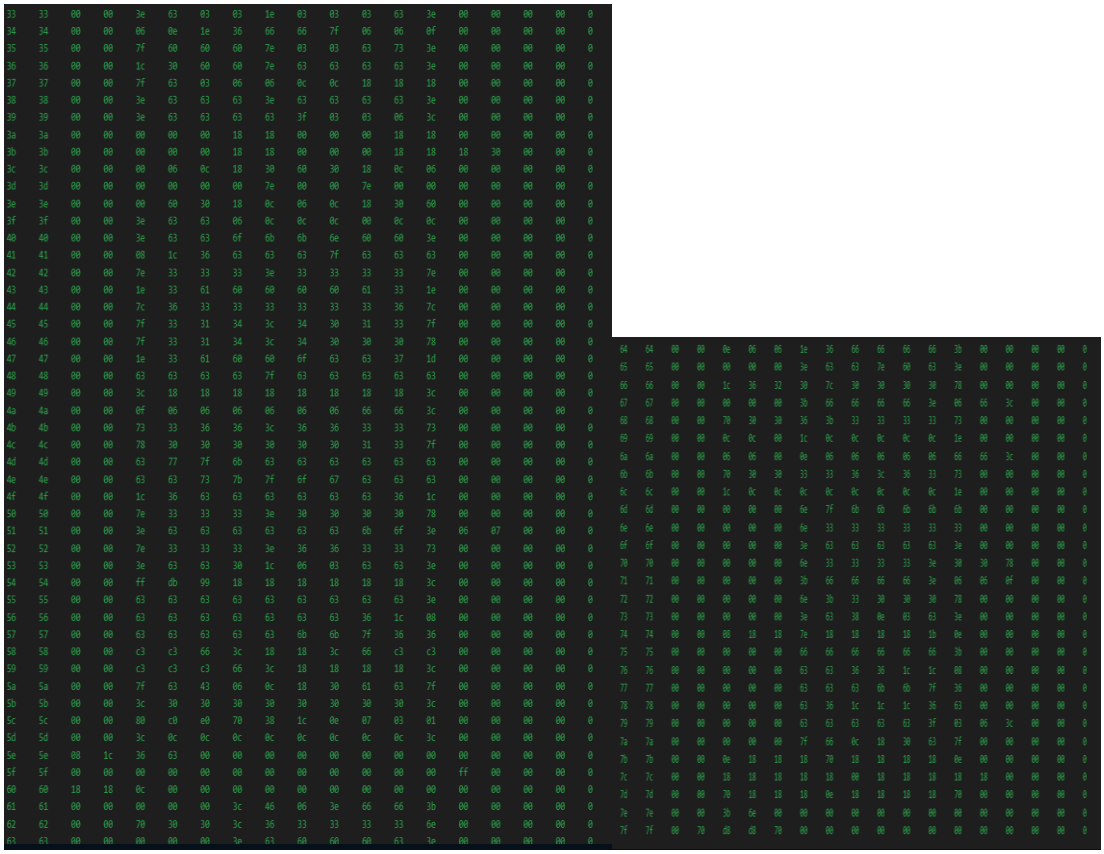


仿真测试图：



功能表：

Cnt	ASCIICOrow1	row2	row3	row4	row5	row6	row7	row8	row9	row10	row11	row12	row13	row14	row15	row16	CodeEr
00	00	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
01	01	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
02	02	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
03	03	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
04	04	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
05	05	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
06	06	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
07	07	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
08	08	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
09	09	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
0a	0a	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
0b	0b	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
0c	0c	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
0d	0d	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
0e	0e	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
0f	0f	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
10	10	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
11	11	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
12	12	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
13	13	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
14	14	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
15	15	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
16	16	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
17	17	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
18	18	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
19	19	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
1a	1a	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
1b	1b	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
1c	1c	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
1d	1d	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
1e	1e	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
1f	1f	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	1 Error!
20	20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0
21	21	00	00	18	3c	3c	3c	18	18	18	00	18	18	00	00	00	0
22	22	00	63	63	22	00	00	00	00	00	00	00	00	00	00	00	0
23	23	00	00	00	36	36	7f	36	36	36	7f	36	36	00	00	00	0
24	24	0c	0c	3e	63	61	60	3e	03	03	43	63	3e	0c	0c	00	0
25	25	00	00	00	00	00	61	63	06	0c	18	33	63	00	00	00	0
26	26	00	00	00	1c	36	36	1c	3b	6e	66	66	3b	00	00	00	0
27	27	00	30	30	30	60	00	00	00	00	00	00	00	00	00	00	0
28	28	00	00	0c	18	18	30	30	30	30	18	18	0c	00	00	00	0
29	29	00	00	18	0c	0c	06	06	06	06	0c	0c	18	00	00	00	0
2a	2a	00	00	00	00	42	66	3c	ff	3c	66	42	00	00	00	00	0
2b	2b	00	00	00	00	18	18	18	ff	18	18	18	00	00	00	00	0
2c	2c	00	00	00	00	00	00	00	00	00	00	18	18	30	00	00	0
2d	2d	00	00	00	00	00	00	00	ff	00	00	00	00	00	00	00	0
2e	2e	00	00	00	00	00	00	00	00	00	00	18	18	00	00	00	0
2f	2f	00	00	01	03	07	0e	1c	38	70	e0	c0	80	00	00	00	0
30	30	00	00	3e	63	63	63	6b	6b	63	63	63	3e	00	00	00	0
31	31	00	00	0c	1c	3c	0c	0c	0c	0c	0c	0c	3f	00	00	00	0
32	32	00	00	3e	63	03	06	0c	18	30	61	63	7f	00	00	00	0



2、数据存储器实验

整体模块设计：

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

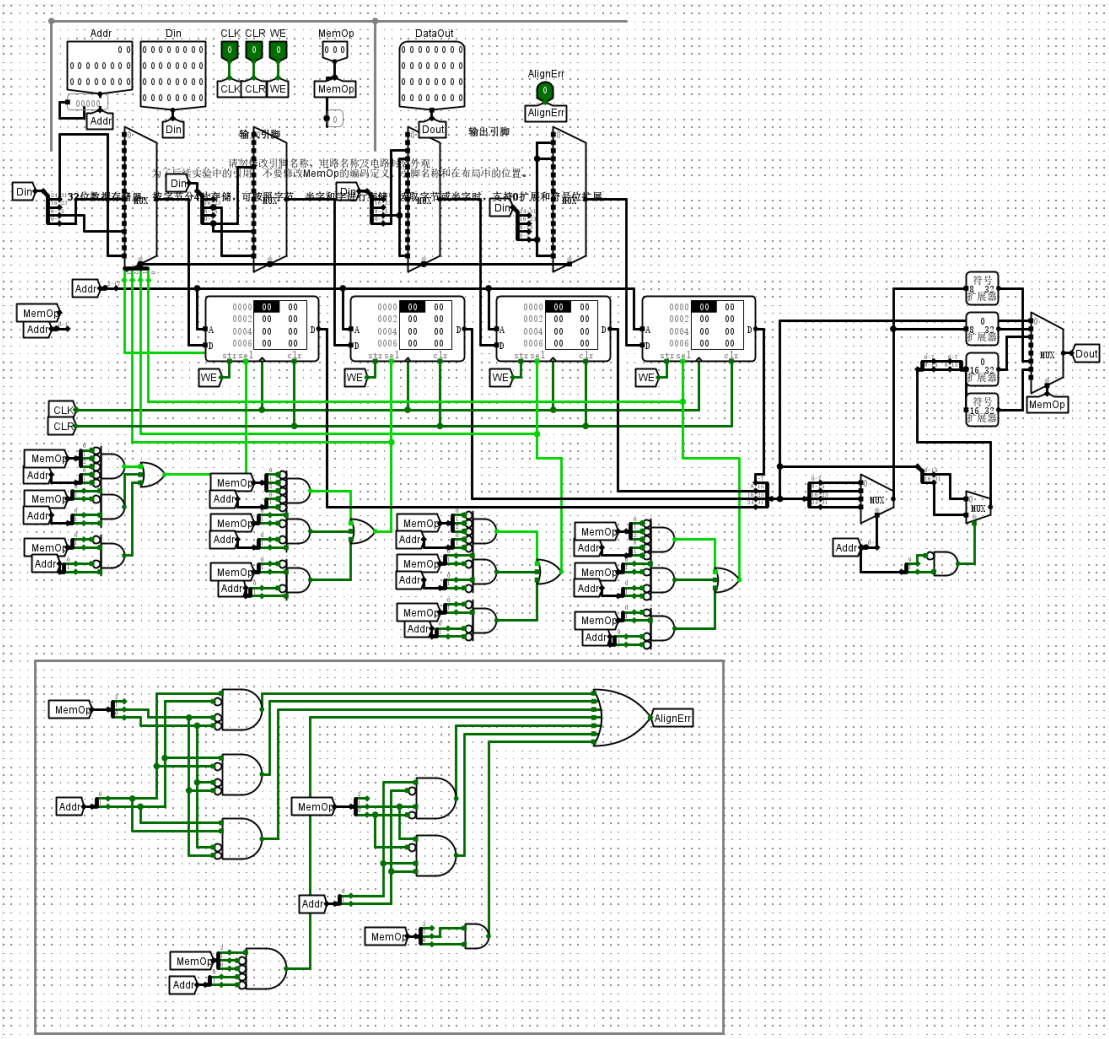
lw 指令从存储器中读取（加载）4 个字节数据到寄存器 rd 中。lh 指令从存储器中加载 2 个字节数据， 根据符号位扩展到 32 位，然后再传送到寄存器 rd 中。lhu 指令从存储器中加载 2 个字节数据，零扩展到 32 位，再传送到寄存器 rd 中。lb 和 lbu 指令的功能类似，只是从存储器中加载 1 个自己数据，然后再根据符 号位或零扩展到 32 位后传送到寄存器 rd 中。 sw、sh 和 sb 指令的功能是将寄存器 rs2 中从低位开始的 4 个字节、2 个字节和 1 个字节的数据存储到 存储器中，此时存储器写使能信号有效。

假设 4 片 RAM3~RAM0 的片选使能端分别定义为 SEL0~SEL3，SEL0 对应最低字节存储器的使能 端。可根据 MemOp 控制信号和存储器地址 Addr 最低 2 位来确定片选信号的数值，当存取多字节数据的有 效地址不是自然对齐时，显示错误信号有效 AlignErr=1。出现未定义的 MemOp 状态时，错误信号也有效 AlignErr=1。

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

原理图及电路图：



输入输出引脚作用：

Addr 表示读取或存取的地址，Din 为存取的数据，MemOp 为控制信号，DataOut 为显示的数据，AlignEr 表示错误信号。

功能表:

Cnt	Addr	WE	CLR	MemOp	Din	Dout	AlignE	
00	00000	1	0	0	00000001	00000000	0	
01	00000	0	0	0	005deece	00000001	0	
02	00010	1	0	5	b61488df	00000000	0	
03	00011	1	0	5	f4111591	00000000	0	
04	00012	1	0	5	023eaf12	00000000	0	
05	00013	1	0	5	b578fa6a	00000000	0	
06	00010	0	0	0	d38a8b1c	6a1291df	0	
07	00100	1	0	6	f5d50649	00000000	0	
08	00102	1	0	6	202e3c08	00000000	0	
09	00100	0	0	0	812fba12	3c080649	0	
0a	00100	0	0	1	6755ebab	00000049	0	
0b	00101	0	0	1	f6e7cab1	00000006	0	
0c	00102	0	0	1	e6bc21b9	00000008	0	
0d	00103	0	0	1	a6c5abc6	0000003c	0	
0e	00100	0	0	2	58228a26	00000649	0	
0f	00102	0	0	2	9dfc1362	00003c08	0	
10	01000	1	0	0	ac5b2754	00000000	0	
11	01000	0	0	5	2f43454c	00000054	0	
12	01001	0	0	5	18c3dc9c	00000027	0	
13	01002	0	0	5	1abbd85c	0000005b	0	
14	01003	0	0	5	cb1b519a	fffffffac	0	
15	01000	0	0	3	0f672c6a	xxxxxxxx	1	Error!
16	01000	0	0	4	e37362d1	xxxxxxxx	1	Error!
17	01000	0	0	7	39eaeb41	xxxxxxxx	1	Error!
18	01001	0	0	6	95c7a81f	xxxxxxxx	1	Error!
19	01003	0	0	6	9b8ab32e	xxxxxxxx	1	Error!
1a	01001	0	0	0	64bb0d7c	xxxxxxxx	1	Error!
1b	01002	0	0	0	72f86d97	xxxxxxxx	1	Error!
1c	01003	0	0	0	52130e8e	xxxxxxxx	1	Error!
1d	01000	0	0	0	1ee9d827	ac5b2754	0	
1e	01000	0	1	0	9056d989	00000000	0	
1f	01000	0	0	0	dc896b07	00000000	0	
20	1f000	1	0	0	913171e3	00000000	0	
21	1f000	0	0	0	6a830aaa	913171e3	0	
22	1f010	1	0	5	ebccdf55	00000000	0	
23	1f011	1	0	5	1bead7a1	00000000	0	
24	1f012	1	0	5	70ce89af	00000000	0	
25	1f013	1	0	5	d6c6f30f	00000000	0	
26	1f010	0	0	0	4f2f15f6	0fafa155	0	
27	1f100	1	0	6	0bfd3217	00000000	0	
28	1f102	1	0	6	74276d58	00000000	0	
29	1f100	0	0	0	c6947ef2	6d583217	0	
2a	1f100	0	0	5	e86e0a71	00000017	0	
2b	1f101	0	0	5	e71009ff	00000032	0	
2c	1f102	0	0	5	d4203d58	00000058	0	
2d	1f103	0	0	5	bd817173	0000006d	0	
2e	1f100	0	0	6	64d7cb03	00003217	0	
2f	1f102	0	0	6	5847aa13	00006d58	0	
30	3fffc	1	0	6	b183207a	00000000	0	
31	3fffe	1	0	6	5333b778	00000000	0	
32	3fffc	0	0	0	fad2465a	b778207a	0	
33	3fffc	0	0	1	608b2671	0000007a	0	
34	3fffc	0	0	5	1229d521	0000007a	0	
35	3fffd	0	0	1	bd93bfe7	00000020	0	
36	3fffd	0	0	5	57945ed9	00000020	0	
37	3fffe	0	0	1	149d0edc	00000078	0	
38	3fffe	0	0	5	d4a71682	00000078	0	
39	3ffff	0	0	1	97e417ee	000000b7	0	
3a	3ffff	0	0	5	2594f1d9	ffffffb7	0	
3b	3fffc	0	0	2	c9e504f9	0000207a	0	
3c	3fffc	0	0	6	e21915e7	0000207a	0	
3d	3fffe	0	0	2	794e8b3c	0000b778	0	
3e	3fffe	0	0	6	2c9b12e4	ffffb778	0	
3f	3fffc	0	0	0	71947694	b778207a	0	

3、取指令部件实验

整体模块设计：

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
B	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
J	imm[20|10:1|11|19:12]					rd	opcode

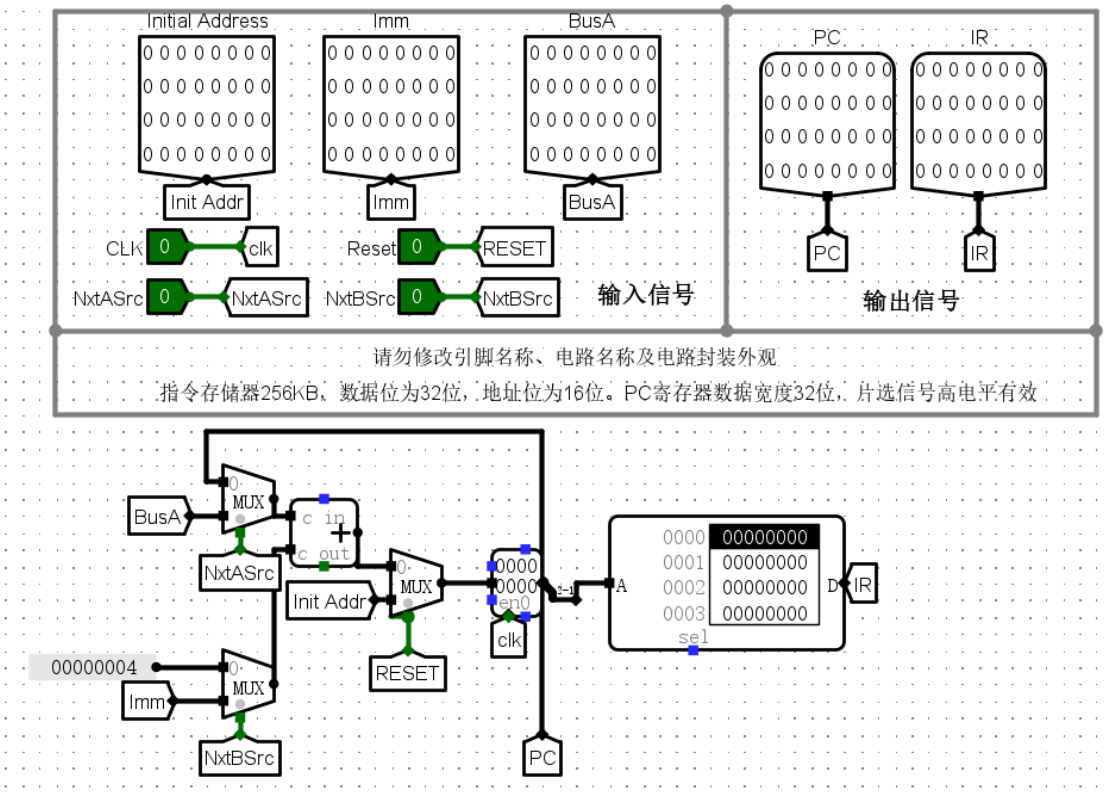
图 5.6　RISC-V 指令格式

在程序执行过程中，下一条指令地址的计算有多种情形：1、顺序执行指令，则 PC=PC+4。2、 无条件跳转指令，jal 指令，PC = PC + 立即数 imm；jalr 指令，PC = R[rs1] + 立即数 imm。3、分支 转移指令，根据比较运算的结果和 Zero 标志位来判断，如果条件成立则 PC = PC +立即数 imm，否则 PC=PC+4。

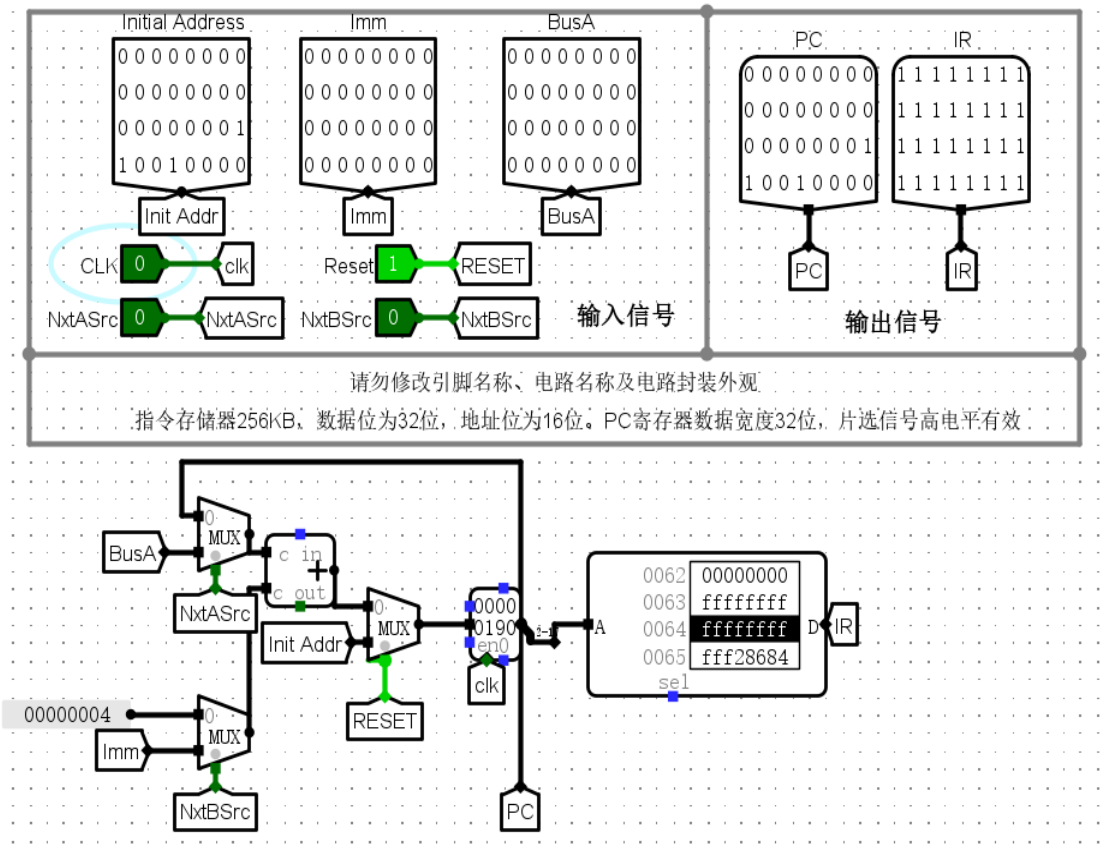
表 5.3 NxtASrc 和 NxtBSrc 赋值定义表

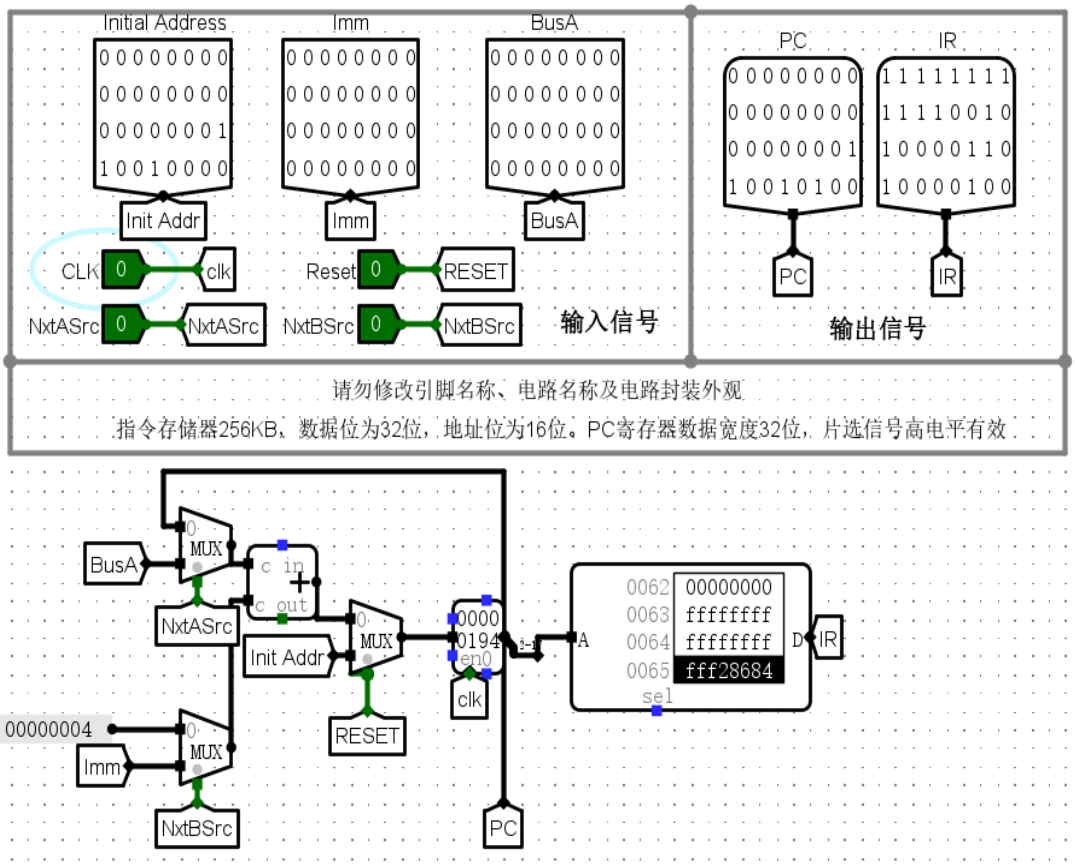
指令类型	NxtASrc	NxtBSrc	下条指令地址
顺序指令	0	0	PC=PC+4
无条件跳转 jal	0	1	PC=PC+Imm
无条件跳转 jalr	1	1	PC= R[rs1]+Imm
分支跳转指令	0	1	条件成立 PC=PC+Imm，否则 PC=PC+4

电路图：



仿真测试图：在在 ROM 存储器存储单元 0064 处写入：ffffffff,fff28684 等数据设置初始地址数据，进行仿真实验





功能表:

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

4、取操作数部件 IDU 实验

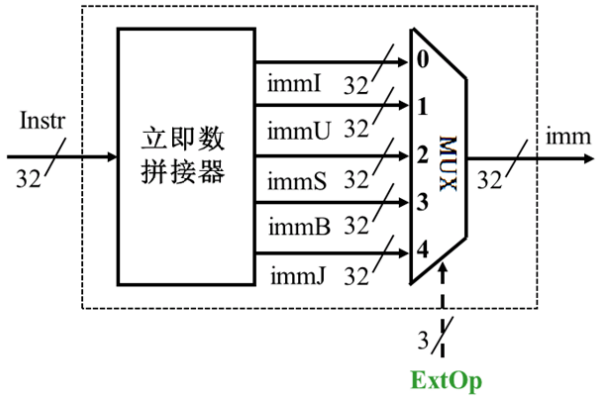
整体模块设计:

ALUAsrc 和 ALUBsrc 用来控制两个多路选择器的输出数据， ALUASsrc 控制 1 个两路选择 器，当 ALUASsrc=0 时，选择 BusA 输出到 ALU 的操作数 A 口，当 ALUASsrc=1 时输出 PC。ALUBSrc 控制 1 个 四路选择器，当 ALUBSsrc=00 时选择 BusB 输出到 ALU 的操作数 B 口，当 ALUBSsrc=01 时选择输出常数 4，当 ALUBSsrc=10 时选择输出 32 位立即数 Imm。

5 种指令的立 即数扩展格式如下:

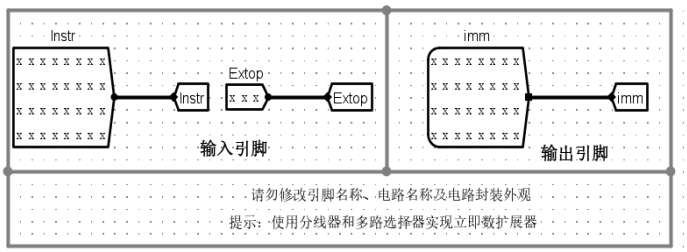
```
immI = {20{Instr[31]}, Instr[31:20]};
immU = {Instr[31:12], 12'b0};
immS = {20{Instr[31]}, Instr[31:25], Instr[11:7]};
immB = {19{Instr[31]}, Instr[7], Instr[30:25], Instr[11:8], 1'b0};
immJ = {11{Instr[31]}, Instr[19:12], Instr[20], Instr[30:21], 1'b0};
```

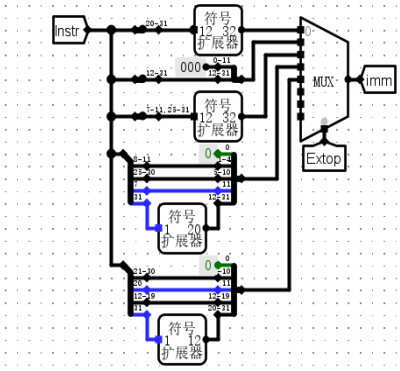
其设计示意图如下所示，通过控制信号 ExtOp 来选择不同立即数编码类型以及在扩展器中进行的扩展操作:

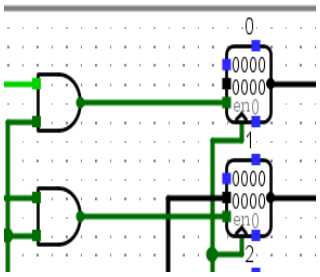


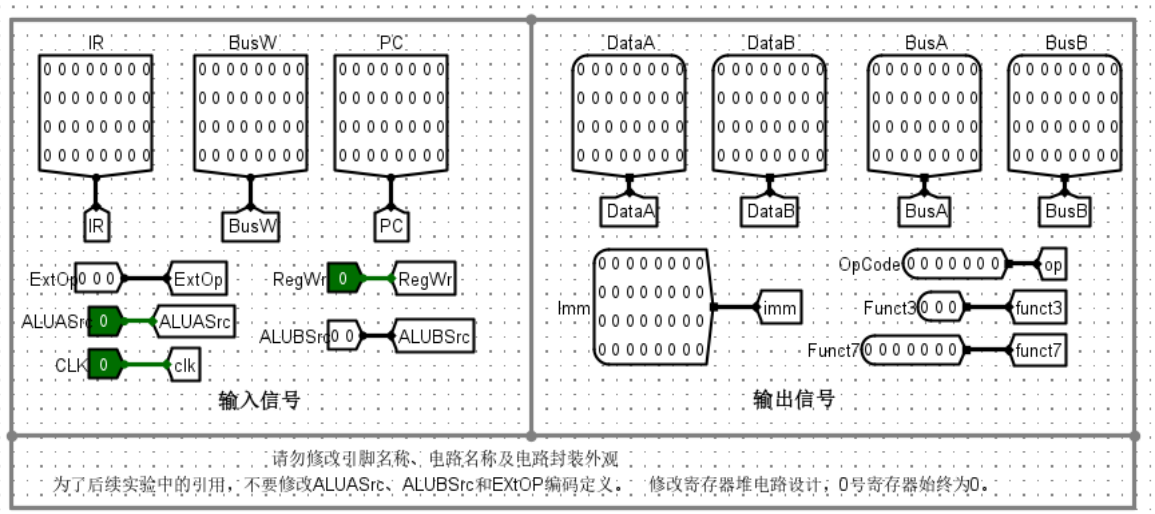
在 RSIC-V 体系架构中，要求 0 号寄存器硬设计为零，因此需要修改寄存器堆子电路，使得 0 号寄存器始终为 0，设 置寄存器堆中的寄存器为上升沿触发。

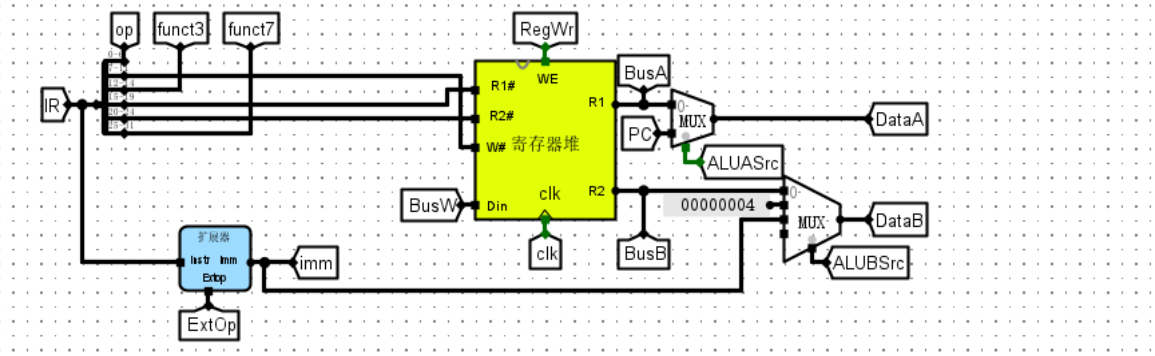
电路图:（立即数扩展器 以及断开 零号寄存器的连线）



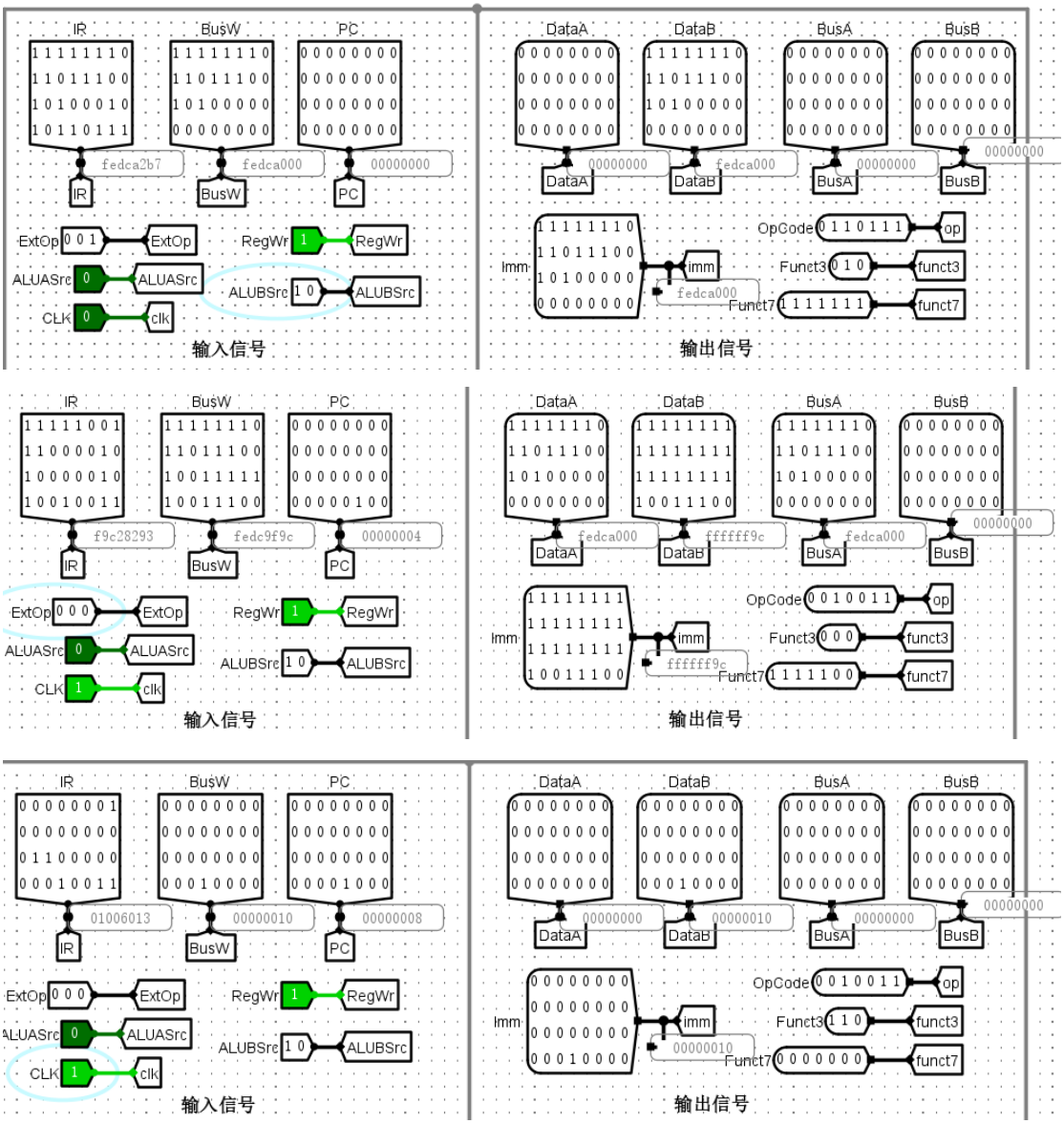








仿真测试图：参照功能表 00，01，02



功能表：

Cnt	IR	BusW	PC	ExtOp	RegWr	ALUASr	ALUBSr	DataA	DataB	BusA	BusB	Imm	OpCode	Funct3	Funct7
00	fedca2b7	fedca000	00000000	1	1	0	2	00000000	fedca000	00000000	00000000	fedca000	37	2	7f
01	f9c28293	fedc9f9c	00000004	0	1	0	2	fedca000	ffffff9c	fedca000	00000000	ffffff9c	13	0	7c
02	01006013	00000010	00000008	0	1	0	2	00000000	00000010	00000000	00000000	00000010	13	6	00
03	06502223	00000064	0000000c	2	0	0	2	00000000	00000064	00000000	fedc9f9c	00000064	23	2	03
04	06400303	ffffff9c	00000010	0	1	0	2	00000000	00000064	00000000	00000000	00000064	03	0	03
05	06601423	00000068	00000014	2	0	0	2	00000000	00000068	00000000	ffffff9c	00000068	23	1	03
06	06405383	00009f9c	00000018	0	1	0	2	00000000	00000064	00000000	00000000	00000064	03	5	03
07	06701623	0000006c	0000001c	2	0	0	2	00000000	0000006c	00000000	00009f9c	0000006c	23	1	03
08	4042d413	ffedc9f9	00000020	0	1	0	2	fedc9f9c	00000404	fedc9f9c	00000000	00000404	13	5	20
09	006444b3	00123665	00000024	0	1	0	0	ffedc9f9	ffffff9c	ffedc9f9	ffffff9c	00000006	33	4	00
0a	00649533	50000000	00000028	0	1	0	0	00123665	ffffff9c	00123665	ffffff9c	00000006	33	1	00
0b	008505b3	4fedc9f9	0000002c	0	1	0	0	50000000	ffedc9f9	50000000	ffedc9f9	00000008	33	0	00
0c	00b2a633	00000001	00000030	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c	4fedc9f9	0000000b	33	2	00
0d	00b2b6b3	00000000	00000034	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c	4fedc9f9	0000000b	33	3	00
0e	40b287b3	aeeed5a3	00000038	0	1	0	0	fedc9f9c	4fedc9f9	fedc9f9c	4fedc9f9	0000040b	33	0	20
0f	06f02823	00000070	0000003c	2	0	0	2	00000000	00000070	00000000	aeeed5a3	00000070	23	2	03
10	0067c263	00000001	00000040	3	0	0	0	aeeed5a3	ffffff9c	aeeed5a3	ffffff9c	00000004	63	4	00
11	0067d263	00000001	00000044	3	0	0	0	aeeed5a3	ffffff9c	aeeed5a3	ffffff9c	00000004	63	5	00
12	0040086f	0000004c	00000048	4	1	1	1	00000048	00000004	00000000	00000000	00000004	6f	0	00
13	004808e7	00000050	0000004c	0	1	1	1	0000004c	00000004	0000004c	00000000	00000004	67	0	00
14	00001917	00001050	00000050	1	1	1	2	00000050	00001000	00000000	00000000	00001000	17	1	00
15	00000000	00000000	00000000	0	0	0	0	00000000	00000000	00000000	00000000	00000000	00	0	00

5、数据通路实验

整体模块设计：

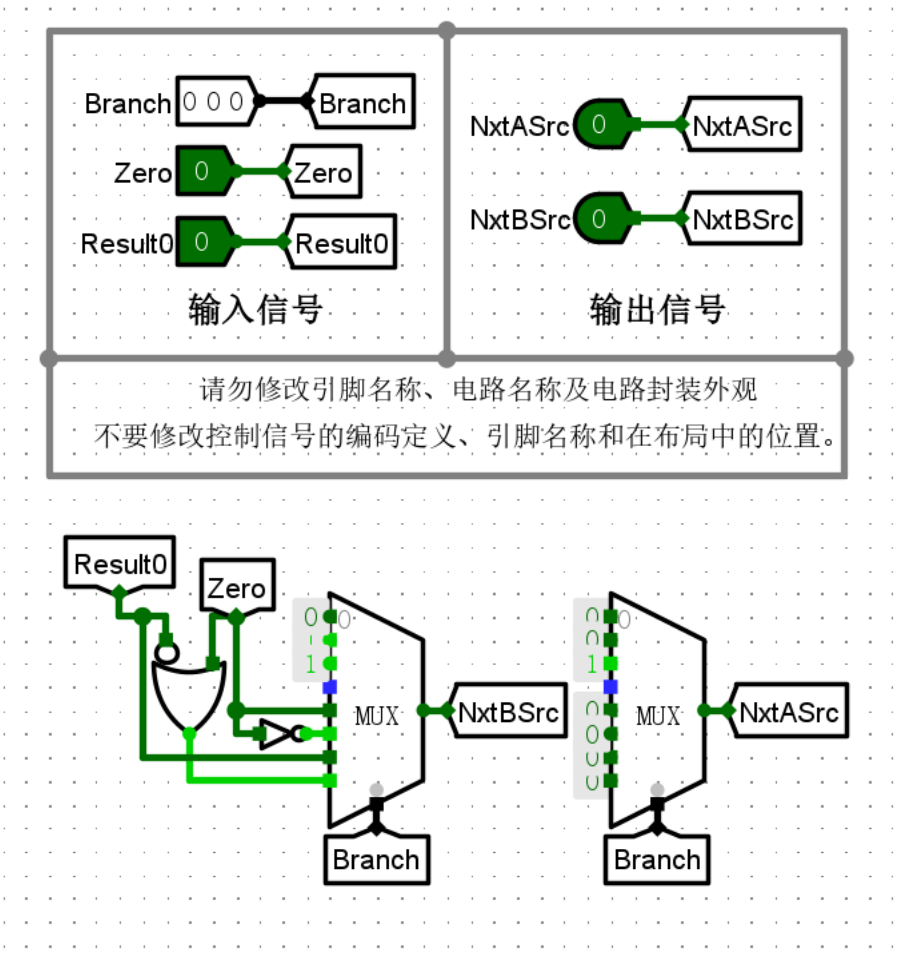
大致可分为取指令 IFU、 取操作数 IDU、执行指令 EX、访问存储器 M 和写回寄存器堆 WB 等阶段。

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。

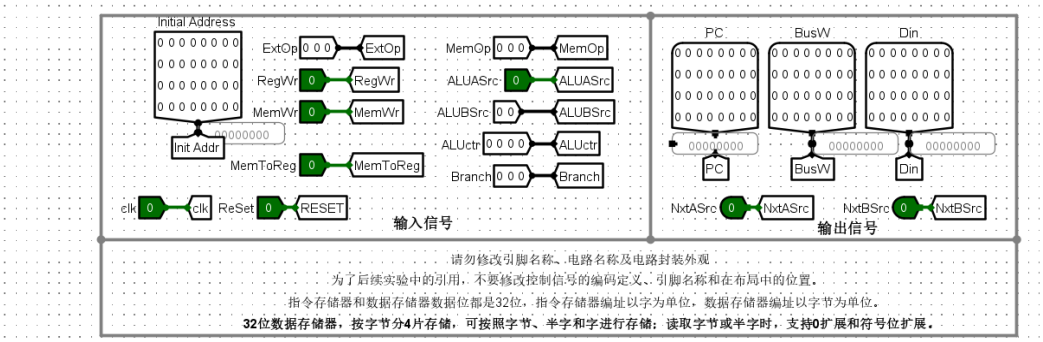
表 5.4 Branch 控制信号含义

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

电路图：



仿真测试图：参照数据表 00，01，02

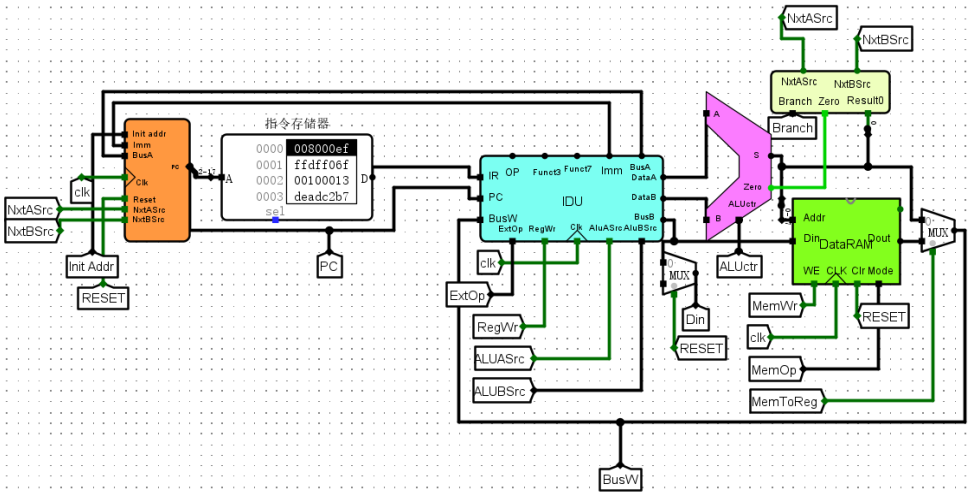


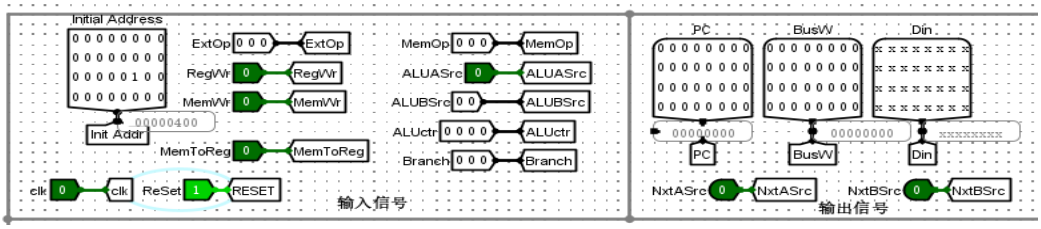
请勿修改引脚名称、电路名称及电路封装外观

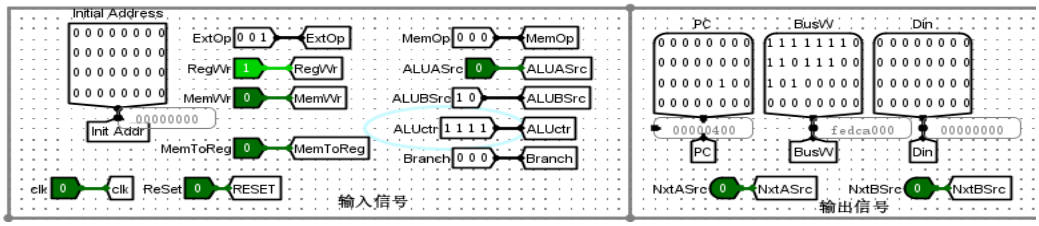
为了后续实验中的引用，不要修改控制信号的编码定义、引脚名称和在布局中的位置。

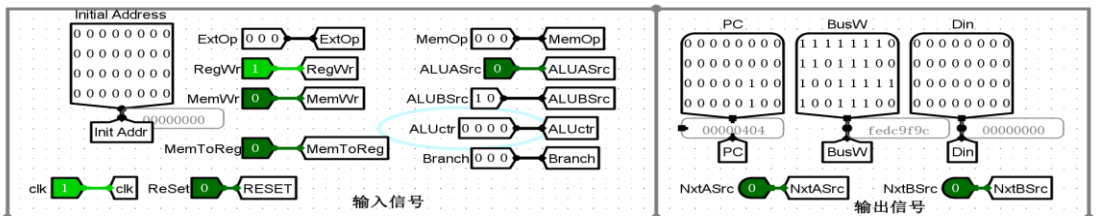
指令存储器和数据存储器数据位都是32位，指令存储器编址以字为单位，数据存储器编址以字节为单位。

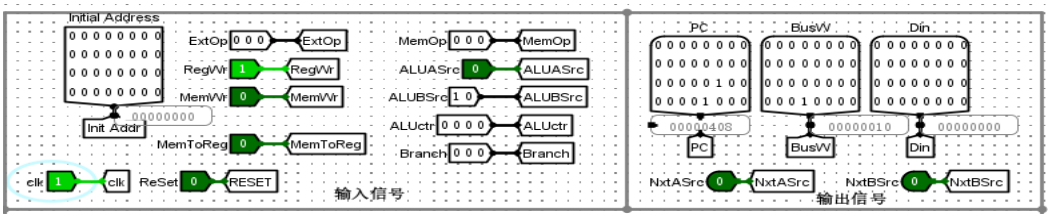
32位数据存储器，按字节分4片存储，可按照字节、半字和字进行存储；读取字节或半字时，支持0扩展和符号位扩展。











数据表：

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

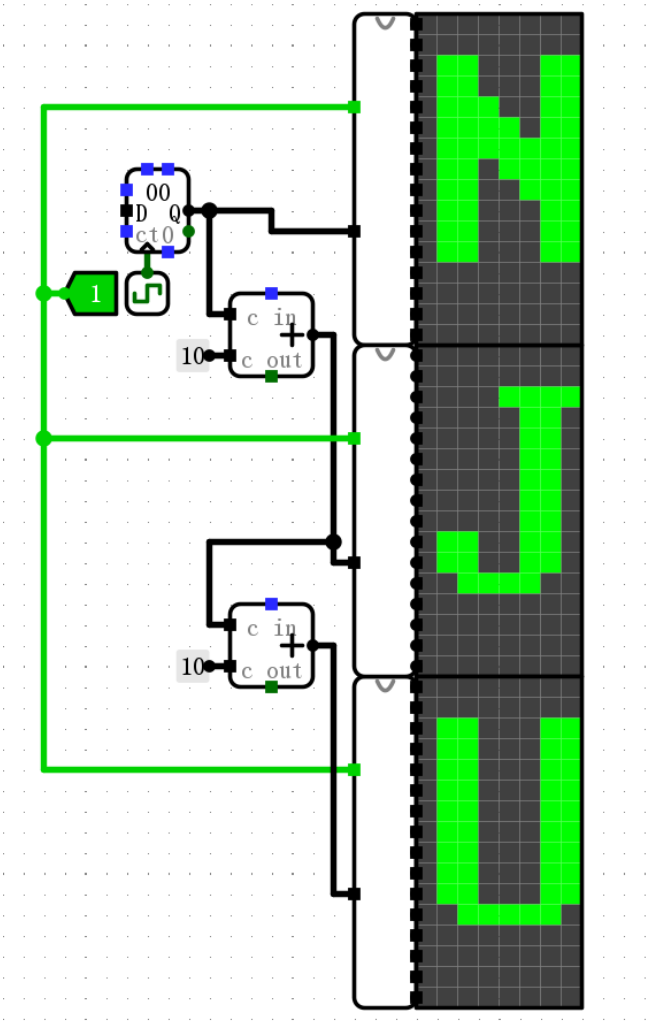
四、 思考题

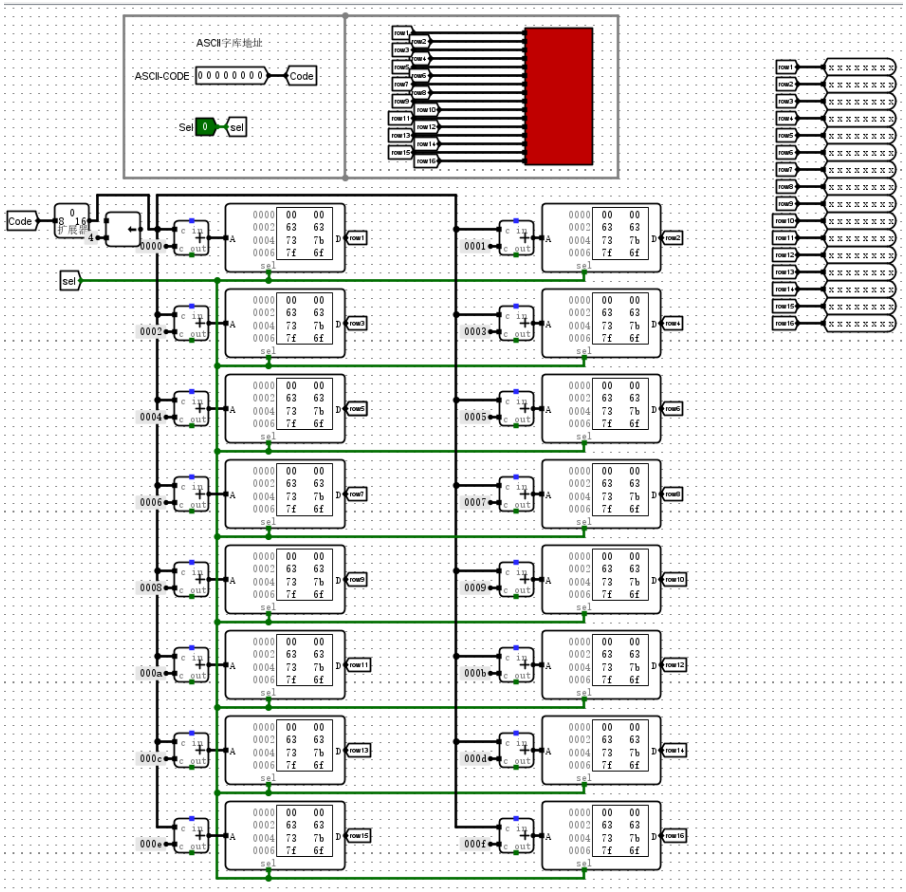
1、 如何利用 ROM 实验实现滚动显示的功能，在 3 个 LED 点阵矩阵中，左右滚动显示 5 个 ASCII 字 符，如“NJUCS”。

整体模块设计：

仿照实验 1，通过编辑镜像数据，将 NJUCS 的 ASCII 码值预先存好，通过寄存器控制 led 点阵每行的值，控制显示的字符，从而实现跑马灯效果。

电路图：





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?

A．寄存器堆在下降沿触发写入，而 PC 寄存器和数据存储器在上升沿触发写入。这种设计导致寄存器堆的数据更新和 PC 寄存器、数据存储器的数据更新在同一个时钟周期内不同步。。

B．如果指令执行过程中需要依赖前一条指令的结果，由于寄存器堆和 PC 寄存器、数据存储器的写入触发沿不同，可能会导致数据传输的时序问题。在一个时钟周期内，寄存器堆的写入发生在 PC 寄存器和数据存储器写入之后，这可能导致在某些情况下，下一条指令读取到的仍然是旧的数据。

C．这种不同触发沿的设计可能会引入额外的延迟。在流水线处理器中，这种不同步的写入触发沿会增加流水线控制的复杂性，可能需要额外的逻辑来处理数据传输和控制信号的同步

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?

例如在 ifu（取指令部件）加入一个结束的控制信号，这样就可以使 cpu 停止取指令。

网上查得:

执行完程序后置入 reset 重置信号，返回初始状态。

在执行完程序之后，进入无限循环状态，不执行新的实际操作。

五、错误现象及分析

实验连线较多，一开始经常会出现线路交叉的情况，后来注意观察线路有没有圆圈，且在线路交叉的地方预留多一点的空间，线路就会变得整洁了很多，出错的情况也少多了。