**北京科技大学实验报告**

学院：计算机与通信工程学院 专业：计算机科学与技术 班级：计173

姓名： 张宝丰 学号：41724081 实验日期：2019年12月20日

**实验名称：**创新设计——利用LED点阵实现生命游戏（John Conway’s Game of life）

**实验目的：**学习各接口芯片功能及应用特性，能够组合多个接口芯片，完成设计实验。

**实验要求：**根据已学习各接口芯片的应用特性，参考本讲义前述实验接线图及代码，自己设计完成综合设计实验。并通过现场验收后，完成本实验报告。

**坚决杜绝抄袭！**

**实验环境：**

操作系统：Windows 10

仿真软件：Proteus 8.6

硬件设备：CZ-CIUS型开放式微机接口实验系统

**实验内容：**

根据已学习各接口芯片的应用特性，参考本讲义前述实验接线图及代码，自己设计完成综合设计实验。要求所选多个芯片或模块的**加权值总和≥3**，权值列表见实验讲义的表3-1。并且，**所选芯片或模块中必须包含8259或8254芯片的其中一个作为功能模块部分**。

**特别要求：**

1、2019年创新实验不得设计**交通灯、电子琴/音乐盒、简单流水灯/跑马灯/流水式霓虹灯、抢答器、投票器**。

2、**至少要在实验箱上验证实现多模块创新设计**。

3、同时实现实验箱设计与Proteus仿真设计，获得相应分数，如果仅实现实验箱设计，则无法得到仿真设计部分的分数。

4、**不能是第二章验证实验中代码的简单叠加**，需要包含一定量自己编写的汇编代码，在验收时需说明自己所编写的代码量大概是多少。

5、代码逻辑具备一定的复杂度，可以折抵1分加权分，但在验收时，需要提供程序流程图。

**实验结果与分析：**

1. **设计应用场景及实现功能**

- 生命游戏简介：

生命游戏（Game of life）是一个元胞自动机（Cellular Automaton），由英国数学家John Horton Conway在1970年发明。这款游戏由一些细胞（网格）构成，每个细胞在下一时刻的状态由它周围的八个细胞所决定。玩家所需要做的只是设定细胞的初始图案，之后便可静静观察多样的演化过程，一些进阶的玩家也会创造出具有某些特性的图案，比如滑翔者（Glider），轻量级宇宙飞船（Light Spaceship）等。

- 游戏规则：

细胞的演化由下面4条规则决定

1. “人口过少”：任何活细胞如果活邻居少于2个，则死掉。

2. “正常”：任何活细胞如果活邻居为2个或3个，则继续活。

3. “人口过多”：任何活细胞如果活邻居大于3个，则死掉。

4. “繁殖”：任何死细胞如果活邻居正好是3个，则活过来。

- 控制方式：

玩家可以通过修改程序中load\_pattern函数来加载不同的预置图案，或者直接修改数据段中的buffer来生成自己想要的图案。

- 其它说明：

受制于硬件，本次实验我只实现了在8\*8 LED点阵上的生命游戏。所有超出网格范围的细胞将不被考虑。

1. **设计思路**

- 软件层面：

数据结构：

利用8个字节作为屏幕缓冲区（buffer），存放当前画面中所有网格的状态，每一位代表一个网格的状态，正好可以满足8\*8点阵的需要。

程序流程：

初始化->显示图案->每隔一段时间刷新图案，并重设8254计数器1的计数初值。

- 硬件层面：

8086芯片是驱动核心；

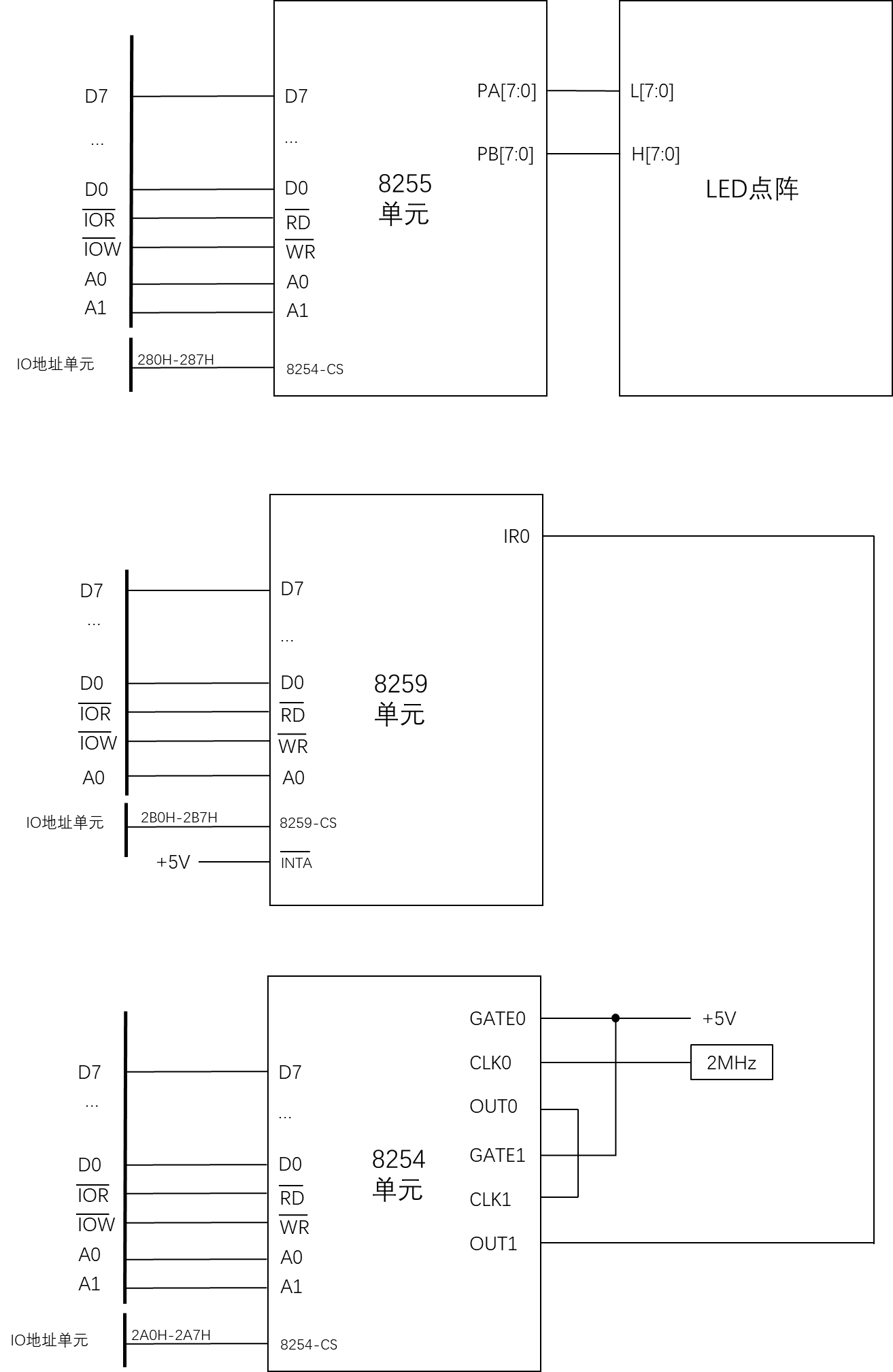
8254计时器实现对2MHz信号的分频，输出1Hz的中断信号：8254的计数器0工作在方式3，输出方波；计数器1工作在方式0，输出中断信号；

8259用于接收8254给出的中断信号，和8086相连实现查询中断，控制字的设置要求：边沿触发、单片8259、需要ICW4。

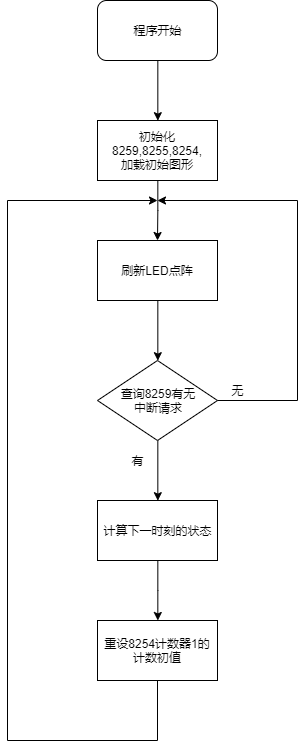
8255芯片用于输出控制LED点阵的信号，端口A和B均为输出，工作在方式0，逐行扫描。

LED点阵接收控制信号，并显示图案，工作在独立模式。

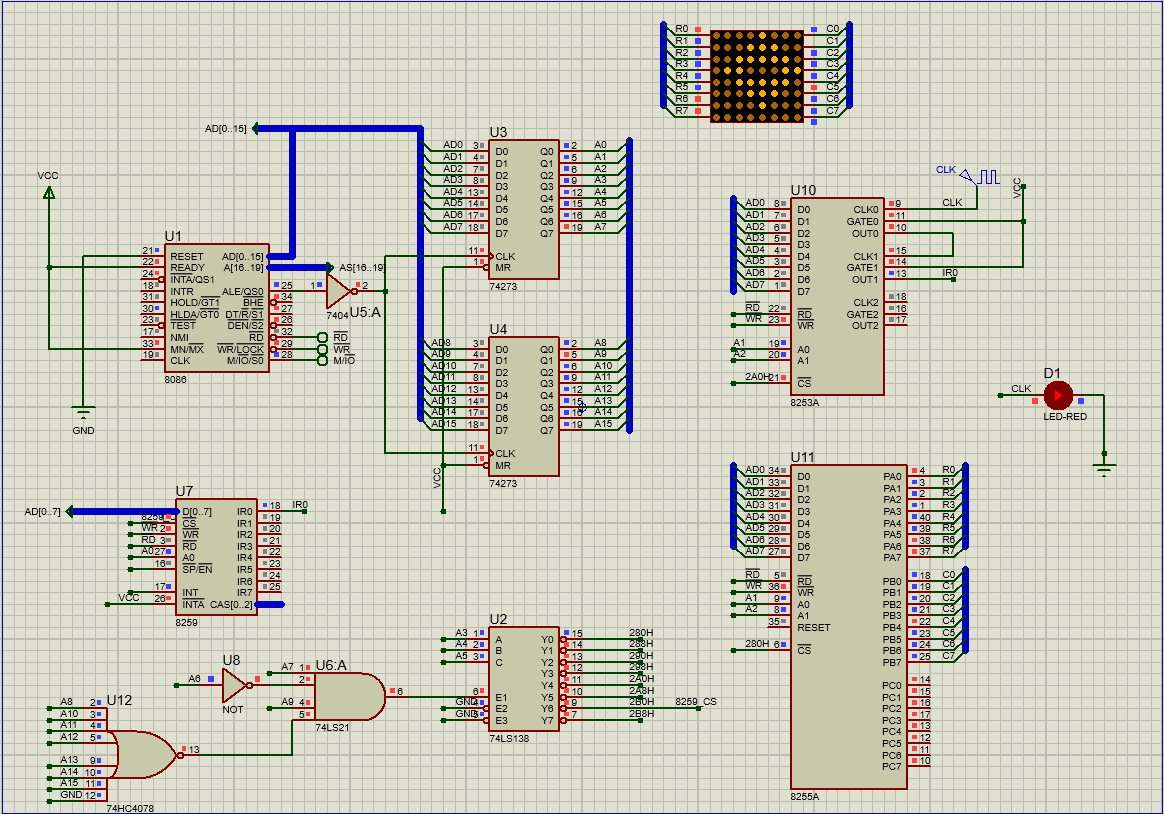
1. **设计接线图**

****

1. **程序流程图**



1. **仿真设计实现**

**设计图：**

**源代码：**与实验源代码类似，请参见6）实验源代码。仿真与实验箱代码的不同处已在注释处标明。

**仿真运行结果：**

1. 滑翔者（Glider）演化图案：



2. 扩散者（Exploder）演化图案：



**仿真设计与实验箱设计的差异：**

经探索与询问与查阅资料，对于中断的处理，Proteus的仿真支持中断向量，而实验箱只支持查询中断，故仿真和实验箱的代码会略有不同。

1. **实验源代码**

IO8255\_MODE EQU 286H

IO8255\_A EQU 280H

IO8255\_B EQU 282H

IO8255\_C EQU 284H

IO8254\_MODE EQU 2A6H

IO8254\_0 EQU 2A0H

IO8254\_1 EQU 2A2H

IO8254\_2 EQU 2A4H

I8259\_1 EQU 2B4H ; 8259的ICW1端口地址

I8259\_2 EQU 2B3H ; 8259的ICW2端口地址

I8259\_3 EQU 2B6H ; 8259的ICW3端口地址

I8259\_4 EQU 2B5H ; 8259的ICW4端口地址

O8259\_1 EQU 2B1H ; 8259的OCW1端口地址

O8259\_2 EQU 2B0H ; 8259的OCW2端口地址

O8259\_3 EQU 2B2H ; 8259的OCW3端口地址

; 每个点由8个字节表示

ROW EQU IO8255\_B

COL EQU IO8255\_A

DATA SEGMENT 'DATA'

buffer DB 00000000B

DB 00000000B

DB 00000000B

DB 00000000B

DB 00000000B

DB 00000000B

DB 00000000B

DB 00000000B

glider DB 00000000B

DB 00100000B

DB 00010000B

DB 01110000B

DB 00000000B

DB 00000000B

DB 00000000B

DB 00000000B

exploder DB 00000000B

DB 00000000B

DB 00001000B

DB 00011100B

DB 00010100B

DB 00001000B

DB 00000000B

DB 00000000B

temp DB 00000000B

DB 00000000B

DB 00000000B

DB 00001100B

DB 00011110B

DB 00001100B

DB 00000000B

DB 00000000B

DATA ENDS

MYSTACK SEGMENT 'STACK'

dw 128 dup (0)

MYSTACK ENDS

CODE SEGMENT 'CODE'

ASSUME CS:CODE, DS:DATA, SS:MYSTACK

START:

MOV AX, DATA

MOV DS, AX

; 初始化8259

MOV DX, I8259\_1 ;初始化8259的ICW1

MOV AL, 00010011b ;边沿触发、单片8259、需要ICW4

OUT DX,AL

MOV DX,I8259\_2 ;初始化8259的ICW2

MOV AL,0B0H

OUT DX,AL

MOV AL,03H

OUT DX,AL

MOV DX, O8259\_1 ;初始化8259的中断屏蔽操作命令字OCW1

MOV AL, 00H ;打开屏蔽位

OUT DX,AL

; 初始化8255: A、B输出

MOV DX, IO8255\_MODE

MOV AL, 80H

OUT DX, AL

MOV DX,ROW

MOV AL,03H

OUT DX,AL

MOV DX,COL

MOV AL,0F1H

OUT DX,AL

; \*\*\*\*\*\*\*\* 仿真用代码 \*\*\*\*\*\*\*\*\*

; 初始化8254工作模式

; 计数器0，方式3

MOV DX, IO8254\_MODE

MOV AL, 00010000B

OUT DX, AL

MOV DX, IO8254\_0

MOV AL, 05H

OUT DX, AL

MOV DX, IO8254\_0

MOV AL, 03H

OUT DX, AL

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; \*\*\*\*\*\*\*\* 上板用代码 \*\*\*\*\*\*\*\*\*

; 初始化8254工作模式

; 计数器0，方式3

MOV DX, IO8254\_MODE

MOV AL, 00110110B

OUT DX, AL

MOV DX, IO8254\_0

MOV AL, 058H

OUT DX, AL

MOV DX, IO8254\_0

MOV AL, 00H

OUT DX, AL

; 计数器1，方式0

MOV DX, IO8254\_MODE

MOV AL, 01110000B

OUT DX, AL

MOV DX, IO8254\_1

MOV AL, 04H

OUT DX, AL

MOV DX, IO8254\_1

MOV AL, 014H

OUT DX, AL

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 加载初始图案

call load\_pattern

; --- 主循环

QUERY:

; 设定每刷新多少次，查询一次中断信号

MOV CX,008FH

DIS\_LOOP:

CALL DISP

LOOP DIS\_LOOP

; 查询中断

MOV DX, O8259\_3 ;向8259发送查询命令(填空）

MOV AL, 0CH ;设置查询方式（填空）

OUT DX,AL

IN AL,DX ;读出查询字

TEST AL,80H ;判断中断是否已响应

JZ QUERY ;没有响应则继续查询

; 如果查询到中断，则刷新画面

call generator

; 设置8254计数器1的计数初值

MOV DX, IO8254\_1

MOV AL, 04H

OUT DX, AL

MOV DX, IO8254\_1

MOV AL, 014H

OUT DX, AL

EOI:

; 向8259发送中断结束命令

MOV DX,O8259\_2

MOV AL,20H

OUT DX,AL

JMP QUERY

ENDLESS:

JMP ENDLESS

; --- 函数：预加载图案

; 设置函数中加载到al的内容可以更改初始图形

load\_pattern proc

push ax

push cx

push si

mov si,0

mov cx,8

load\_l1:

mov al,byte ptr glider[si] ; 加载滑翔者

;mov al, byte ptr exploder[si] ; 加载扩散者

;mov al, byte ptr light\_spaceship[si] ; 加载宇宙飞船

mov byte ptr buffer[si],al

inc si

loop load\_l1

pop si

pop cx

pop ax

ret

load\_pattern endp

; --- 函数：刷新画面

; 对于8\*8的每个格子，调用judge

; 生成的下一时刻状态存储在temp中

; 代码较长，略去的重复部分参见life\_game\_onboard.asm

generator proc ; 刷新画面，更新buffer

mov si,0

mov di,0

call judge

mov si,0

mov di,1

call judge

mov si,0

mov di,2

call judge

mov si,0

mov di,3

call judge

mov si,0

mov di,4

call judge

mov si,0

mov di,5

call judge

mov si,0

mov di,6

call judge

mov si,0

mov di,7

call judge

; 此处省略重复代码，请参见life\_game\_onboard.asm

……

call copy\_to\_buffer

generator endp

; --- 函数DISP：将buffer输出给LED

; 在仿真时，R=1,C=0，对应的点才会亮

; 在上板时，R=1,C=1，对应的点才会亮

; 视上板还是仿真，注释掉程序中的一行代码

DISP PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

MOV CX,8

MOV BL,1

MOV SI,0

DISP\_L1:

MOV DX,ROW

MOV AL,BL

OUT DX,AL

CALL DELAY

MOV DX,COL

MOV AL,[si]

; \*\*\*\*\*\*\*\*\*\* 仿真时注释掉这部分 \*\*\*\*\*\*\*\*\*\*\*\*

;NOT AL

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUT DX,AL

CALL DELAY

SHL BL,1

INC SI

LOOP DISP\_L1

POP SI

POP DX

POP CX

POP BX

POP AX

RET

DISP ENDP

; --- 函数DELAY: 延时子程序

; 调节CX大小可以控制时长

DELAY PROC

PUSH CX

MOV CX,000FH

DL1: LOOP DL1

POP CX

RET

DELAY ENDP

; --- 函数copy\_to\_buffer

; 将temp的内容复制给buffer

copy\_to\_buffer proc

push ax

push cx

push si

mov si,0

mov cx,8

cpt\_l1:

mov al,byte ptr temp[si]

mov byte ptr buffer[si],al

inc si

loop cpt\_l1

pop si

pop cx

pop ax

ret

copy\_to\_buffer endp

; --- 函数judge

; 依据周围8格的值设置temp[si][di]的状态

judge proc

mov ax,0

cmp\_1:

; 左上

cmp si,0

je cmp\_2

cmp di,0

je cmp\_2

push si

push di

dec si

dec di

call get\_buffer

add ax,bx

pop di

pop si

cmp\_2: ; 上

cmp si,0

je cmp\_3

push si

push di

dec si

call get\_buffer

add ax,bx

pop di

pop si

cmp\_3:

; 右上

cmp si,0

je cmp\_4

cmp di,7

je cmp\_4

push si

push di

dec si

inc di

call get\_buffer

add ax,bx

pop di

pop si

cmp\_4:

; 左

cmp di,0

je cmp\_5

push si

push di

dec di

call get\_buffer

add ax,bx

pop di

pop si

cmp\_5:

; 右

cmp di,7

je cmp\_6

push si

push di

inc di

call get\_buffer

add ax,bx

pop di

pop si

cmp\_6:

; 左下

cmp si,7

je cmp\_7

cmp di,0

je cmp\_7

push si

push di

inc si

dec di

call get\_buffer

add ax,bx

pop di

pop si

cmp\_7:

; 下

cmp si,7

je cmp\_8

push si

push di

inc si

call get\_buffer

add ax,bx

pop di

pop si

cmp\_8:

; 右下

cmp di,7

je cmp\_end

cmp si,7

je cmp\_end

push si

push di

inc si

inc di

call get\_buffer

add ax,bx

pop di

pop si

cmp\_end:

; 结束判断，为buffer[si][di]赋值

mov bl,buffer[si] ; bx = buffer[si]

mov cx,7 ; cx = 7-di

sub cx,di

shr bl,cl

and bx,1 ; 当buffer[si][di]为1时，bx = 1

cmp bx,1 ; 判断buffer[si][di]是死、活

jne cmp\_die

cmp\_live:

cmp ax,2

jl is\_die ; 活细胞邻居少于2，死掉

cmp ax,4

jl is\_live ; 活细胞邻居是2或3，存活

jmp is\_die ; 大于等于4个，死掉

cmp\_die:

cmp ax,3 ; 死细胞邻居=3

je is\_live ; 繁殖

jmp is\_die

is\_live:

call set\_temp ; 置1

jmp re\_end

is\_die:

call reset\_temp ; 置0

jmp re\_end

re\_end:

ret

judge endp

; --- 函数get\_buffer

; 获取buffer[si][di]的值，并存放在bx当中

get\_buffer proc

mov bl,buffer[si] ; bx = buffer[si]

mov cx,7 ; cx = 7-di

sub cx,di

shr bl,cl

and bx,1 ; 当buffer[si][di]为1时，bx = 1

ret

get\_buffer endp

; --- 函数set\_temp

; 将temp[si][di]的值设置为1

set\_temp proc

push ax

push bx

push cx

push dx

mov al,temp[si]

mov cx,7

sub cx,di

mov bl,1

shl bl,cl

or al,bl

mov temp[si],al

pop dx

pop cx

pop bx

pop ax

ret

set\_temp endp

; --- 函数reset\_temp

; 将temp[si][di]的值设置为0

reset\_temp proc

push ax

push bx

push cx

push dx

mov al,temp[si]

mov cx,7

sub cx,di

mov bl,1

shl bl,cl

not bl

and al,bl

mov temp[si],al

pop dx

pop cx

pop bx

pop ax

ret

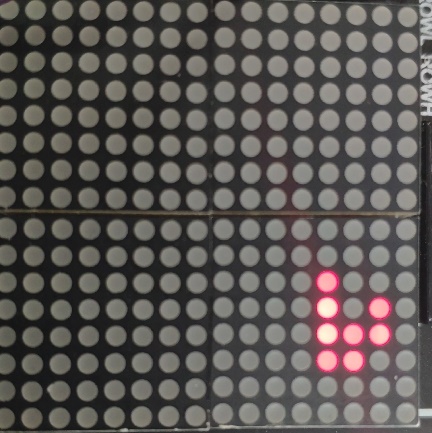
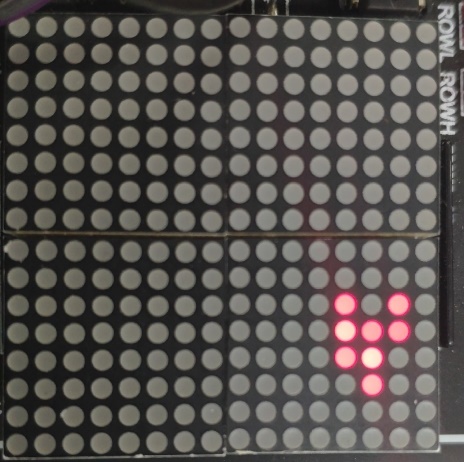
reset\_temp endp

CODE ENDS

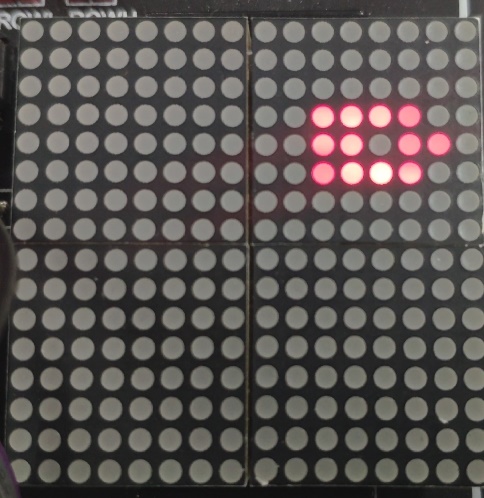
END START

1. **实验现象**

1. 加载滑翔者（Glider）的运行结果：



2. 加载扩散者（Exploder）的运行结果：



以上是设定不同初始状态下的运行效果，基本符合预期，图形按照规则正常扩散。

1. **是否在实现功能的基础上进行了代码的优化或改进**

在代码中，我将许多重复的工作封装为函数，从而减少代码量。如判断下一时刻状态的judge，更新画面的generator，还有set\_temp和reset\_temp等。

1. **实现效果的局限性分析**

实验箱的运行结果和仿真相比有所区别，主要在于一些本来不该亮的LED亮了，猜想可能有下面两个原因：

- LED点阵刷新频率过快，电位信号未来得及切换，致使显示出错；

- 计算下一帧画面的generator函数运行不正确。

因为仿真结果是正确的，第一个原因的可能性会比较大，后续可以尝试增加延时来降低刷新率，或者针对DISP函数进行模块测试，进而排查出原因。

**实验结论（讨论）：**

本次接口实验中，我通过各类参考书、参考代码、老师的悉心指导、同学间交流，以及自行摸索，终于掌握了x86汇编语言的基本用法，并体会了通过汇编语言直接控制硬件的过程，进一步地理解了8086 CPU的工作过程，还有8255并口通信芯片、8254定时器/计数器芯片、8259中断控制芯片的用法，还学会了LED点阵在独立工作模式下的使用，以及128X64 LCD液晶屏的用法，最终制作出8x8 LED点阵上的生命游戏，也收货了一些小小的成就感。总的来说，这次动手实践令我受益良多。

但是这次实验课也有一些可以改进的地方，总结如下：

1. **存在“人为造成的信息不足”**：实验箱对我们来说是一个“黑箱”，我们无法洞知其内部构造，更不知道各个部件有哪些和标准不同的地方，有些给出了英文的说明说，而另一些却没有。同学们在缺乏完整说明书的情况下，只能一点点自行摸索，不断踩坑并填坑，我想这一点有些背离接口实验的初衷。比如我花了很长时间试验实验箱上的LCD液晶屏能否按点阵显示数据，最终发现它只能显示字符，这给我造成了很大的挫败感，我想其他同学也会有类似的体验。建议和厂商协调，拿到实验箱说明书，即便没有，也应当获知实验箱上每个芯片的具体型号，而不是一句通用的中文描述，这样去网络上查找，也能有的放矢。

2. **可以总结一个“常用功能函数库”**：在接口实验中，有一些函数功能模块是通用的，比如延时子程序、初始化子程序、LED点阵显示、液晶屏刷新等，我觉得可以把这些功能模块总结一下，写好注释，说明参数的用途，这既能减少重复的踩坑、减少额外的工作量，也能帮助感兴趣的同学更好理解每个器件的功用，把精力用在发挥自己的创造力上。

3. **个人觉得实验要求中存在着对于同学们的“强烈不信任”**：比如在预习报告中要求同学们对“所用的编辑器画面”截图，这个要求令人匪夷所思；又比如“特别要求”中的每一条，都仿佛充斥着对于同学们满满的不信任感。可能是个人心理过于敏感，也可能是确有其事。虽然只要自己认真做，上面的要求都可以轻松达成，但是我还是建议在提出实验要求的时候，谨慎地斟酌用词，莫因这些打击了积极性，限制了创意与构思。