

Projeto CE4411: Jogo memorização de LEDS

ROSA, Vitor Acosta da
RA: 22.218.006-9

BARBOSA, Andy Silva
RA: 22.218.025-9

14 de Maio de 2020

1 Descrição do projeto

Para o projeto foi escolhido desenvolver um jogo de memorização de LEDS. No qual, foi utilizado o simulador EdSim51 para a criação e execução do código para um microcontrolador 8051.

No jogo, o usuário deve memorizar uma sequência aleatória de LEDS que acendem um de cada vez e apertar os botões na ordem correta para todo LED aceso. Em caso de vitória ou derrota no jogo, o *display* LCD apresentará a mensagem para cada ocorrência, sendo possível começar um novo jogo.

Visite: <https://www.youtube.com/watch?v=Ulcj7i8Uros> para a visualização do funcionamento do projeto.

2 Desenho esquemático

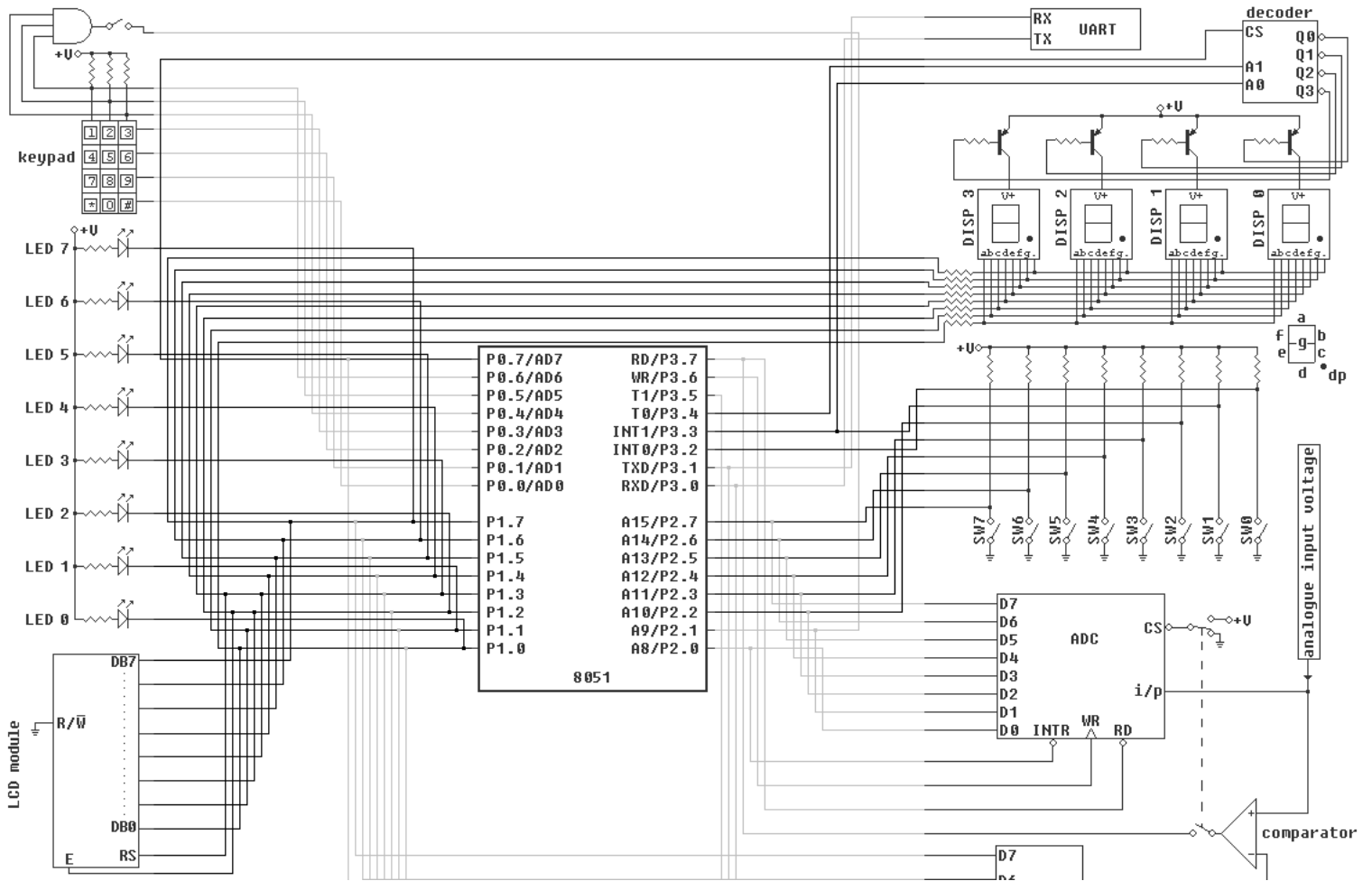


Figure 1: Diagrama esquemático do 8051 para o jogo. Fonte: EdSim51

Para o jogo, como mostra a figura 1, foram utilizados somente: o *display* LCD para exibir a condição de vitória ou derrota, o *LED Bank* para acender ou apagar os LEDs de 7 a 2, o *Switch Bank* para a inserção da sequência pelo usuário e o *Multiplexed 7-Segment Display*.

Vale ressaltar que os pinos P3.2 e P3.3 foram definidos como interrupções externas, e ocupam as chaves SW0 e SW1 respectivamente. Por tal fato, ambas as chaves e os LEDs respectivos 0 e 1, não são utilizados na sequência para o jogo, restando os LEDs de 2 a 7 e as chaves de SW2 a SW7.

3 Diagrama

3.1 Considerações gerais

Para que o jogo flua de maneira correta, e que apresente o melhor resultado e jogabilidade possível, é recomendo jogar com a frequência de *update* igual a 8, o que pode ser configurado facilmente no EdSim51, através da opção **Update Freq.**, como mostra a figura 2.



Figure 2: Configuração geral pré-jogo.

3.2 Utilização das portas

Para o jogo, as portas foram configuradas da seguinte maneira:

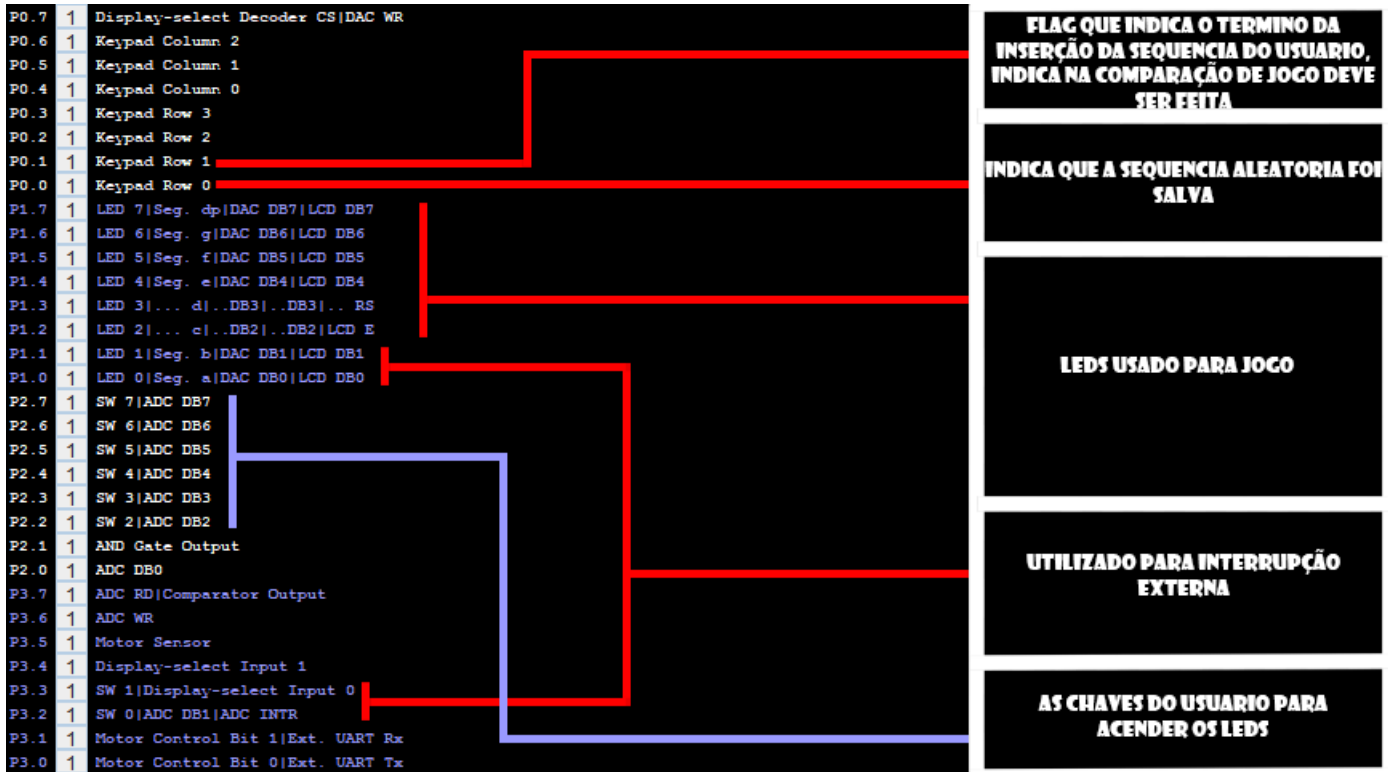


Figure 3: Configuração das portas.

3.3 Uso dos registradores

The screenshot shows the EdSim51 debugger interface. At the top, the System Clock is set to 12.0 MHz. Below this, various registers are displayed in a grid. The PC register is highlighted with a blue box and the value 0x018A. The PSW register is also highlighted with a blue box and the value 0000. The memory dump at the bottom shows the first 80 bytes of memory, with the first 16 bytes (00 to 0F) containing the values 60, 50, 00, 00, 00, 00, 3C, 00, 00, 00, 00, 00, 00, 00, 00, 00.

System Clock (MHz)	12.0	8	Update Freq.
SBUF			
R/O	W/O	TH0	TL0
0x00	0x00	0x00	0x0A
RXD	TXD	TMOD	0x02
1	1	TCON	0x15
SCON	0x00		
pins bits			
0xFF	0xFF	P3	0x00
0xFF	0xFF	P2	0x00
0xFF	0xFF	P1	0x00
0xFF	0xFF	P0	0x00
PC			
0x018A			
PSW			
0000			
R7			
0x00			
R6			
0x3C			
R5			
0x00			
R4			
0x00			
R3			
0x00			
R2			
0x00			
R1			
0x50			
R0			
0x60			
B			
0x00			
ACC			
0x00			
PSW			
0x00			
IP			
0x00			
IE			
0x87			
PCON			
0x00			
DPH			
0x00			
DPL			
0x00			
SP			
0x07			
Modify RAM			
addr 0x00 0x00 value			
Data Memory			
0 1 2 3 4 5 6 7 8 9 A B C D E F			
00 60 50 00 00 00 00 3C 00 00 00 00 00 00 00 00			
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
Copyright ©2005-2016 James Rogers			
Remove All Breakpoints			

Figure 4: Utilização dos registradores. Fonte: EdSim51

Configurações dos registradores, dada por:

R0 (Espaço de memória 00): Ponteiro de memória (com o valor inicial 60) para o salvamento da sequência inserida pelo usuário no decorrer do jogo.

R1 (Espaço de memória: 01): Ponteiro de memória (com valor inicial 50) para o salvamento da sequência aleatória gerada pelo algoritmo.

R6 (Espaço de memória: 06): Utilizado para gerar delay, seu valor padrão é dado por $3C_{16}$ ou 60_{10} .

Os demais registradores não são utilizados na implementação do jogo. Esses valores são os iniciais, *setados* por uma rotina de pré-jogo.

3.4 Números aleatórios

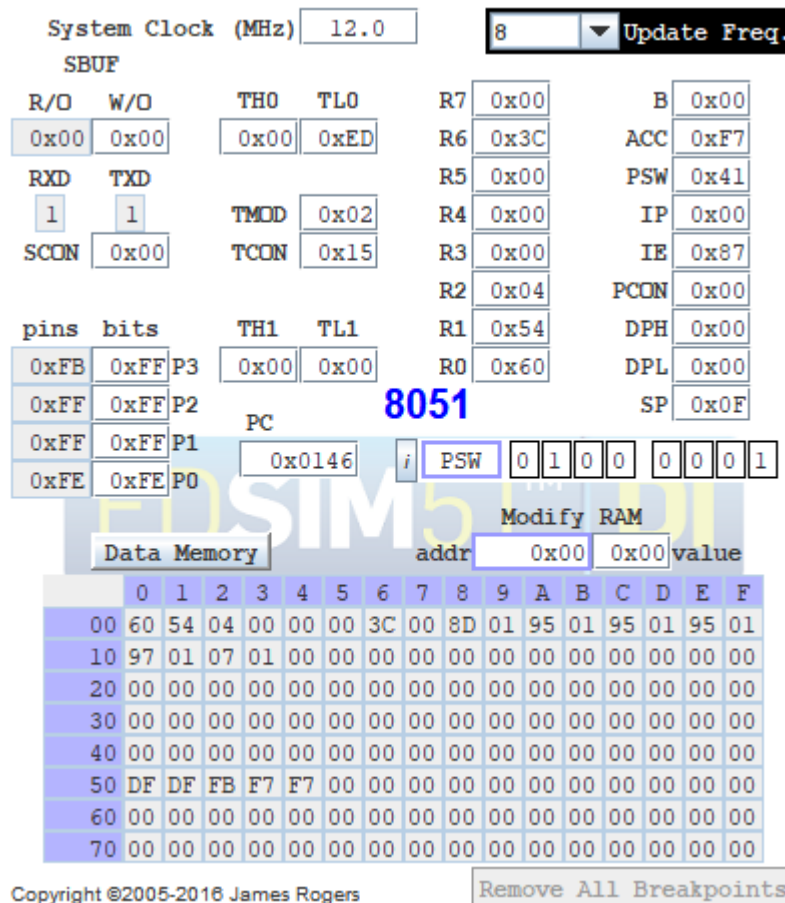


Figure 5: Sequência aleatória de LEDS nas posições de memória 0x50 a 0x54.
Fonte: EdSim51

No espaço de memória entre 0x50 e 0x54 é salvo a sequência aleatória gerada pelo algoritmo.

Tal sequência é gerada em três partes.

Rotina "Gera-seed": A partir do valor disposto no *timer0* (TL0) é calculado um novo valor de recarga para esse *timer*, a fim de quebrar o ciclo e manter a aleatoriedade entre cada LED aceso da sequência.

O cálculo da semente é realizado da seguinte maneira: primeiramente, multiplica-se o valor disposto no TL0 por 17_{10} , em seguida é feita uma rotação de todos os bits para a esquerda, considerando o *carry* nesta operação, como a multiplicação é separada entre MSB (armazenado no registrador B) e LSB (armazenado no

acumulador A), a próxima etapa envolve a soma de ambos e para finalizar há a sobreposição do valor de TL0 por essa semente.

Rotina "Random": Essa rotina gera de fato a sequência aleatória de LED.

Para acender um LED por vez é fornecida uma sequência inicial (dada por 0111111_2 , na qual cada número representa um LED).

Antes de qualquer cálculo, é chamada a rotina **Gera-seed**, após isso é movido o valor do contador para o acumulador A, em seguida, é feita a divisão por 6 (por conta dos LEDS que são utilizados no jogo), o resto (armazenado no registrador B) é considerado, e o quociente (armazenado no acumulador A) descartado.

Dessa forma, o resto representará a quantidade de vezes que a sequência de 8bits iniciais (0111111_2) passará na rotina de rotação.

Rotina "Rotate": Rotaciona para a direita a sequência de 8bits calculada na rotina Random disposta no acumulador A, o número de vezes armazenado no registrador B.

Após todo esse processo, é chamada a rotina "Salva-Seq", que, a partir do ponteiro de memória dado pelo registrador R1, armazena a sequência entre as posições 0x50 e 0x54.

3.5 Números inseridos pelo usuário

System Clock (MHz)

SBUF

R/O	W/O	TH0	TL0	R7	0x00	B	0x00
0x00	0x00	0x00	0x8F	R6	0x3C	ACC	0xF7
RXD	TXD			R5	0x00	PSW	0x41
1	1	TMOD	0x02	R4	0x00	IP	0x00
SCON	0x00	TCON	0x35	R3	0x00	IE	0x87
				R2	0x04	PCON	0x00
pins	bits	TH1	TL1	R1	0x54	DPH	0x00
0xFB	0xFF	P3	0x00	R0	0x64	DPL	0x00
0xFF	0xFF	P2				SP	0x0F
0xFB	0xFB	P1					
0xFE	0xFE	P0					

PC 0x012A **8051**

PSW 0 1 0 0 0 0 0 1

Modify RAM

Data Memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	64	54	04	00	00	00	3C	00	8D	01	95	01	95	01	95	01
10	F3	00	07	01	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	DF	DF	FB	F7	F7	00	00	00	00	00	00	00	00	00	00	00
60	F7	EF	FB	BF	FB	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2016 James Rogers

Figure 6: Sequência de LEDS inserida pelo usuário nas posições de memória 0x60 a 0x64. Fonte: EdSim51

Assim que a sequência aleatória gerada pelo algoritmo é salva nas posições de 0x50 à 0x54, o próximo passo é salvar o *input* do usuário.

O usuário sabe que é a sua vez de inserir a sequência pois o LED 7 fica em modo alternado, acendendo e apagando a cada execução do loop-insert, como mostra a figura 7.

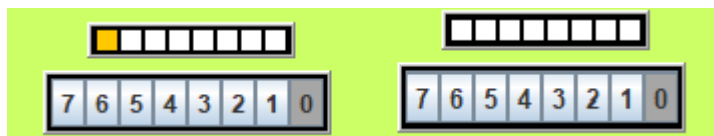


Figure 7: LED 7 alternando estados enquanto o algoritmo espera a inserção do usuário. Fonte: EdSim51

Para a inserção do usuário são envolvidas três rotinas.

Rotina "Loop-insert": Após detectado a necessidade de *input* do usuário para verificar a vitória ou a derrota, é chamada a rotina loop-insert. Ela, é responsável por detectar qualquer alteração de estado entre as chaves de SW7 à SW2, caso alguma seja pressionada, uma segunda rotina é invocada, a "Armazena-user".

Rotina "Armazena-user": A primeira coisa que essa rotina realiza é o acendimento do LED respectivo a chave pressionada. Em seguida, se o ponteiro de memória (dado pelo registrador R0) é válido, é chamada a rotina "Salva-Usr".

Rotina "Salva-Usr": Rotina responsável por armazenar no local de memória dado pelo registrador R0, a chave pressionada pelo usuário.

3.6 Comparação entre sequência aleatória e sequência do usuário

Após realizado todo o processo de geração e salvamento de sequências, tanto aleatória quanto do usuário, o algoritmo tem permissão (através de *Flags*) de comparar ambas.

A rotina responsável por isso é bem simples, haja vista que o algoritmo já conhece onde está salvo as duas sequências através dos registradores R0 e R1. O processo envolve comparação por meio da instrução **CJNE**, na qual, os operandos são os valores dispostos na memória entre 0x50 a 0x54 e 0x60 a 0x64. Nessa perspectiva, os valores são conferidos dois a dois, de modo que o valor presente em 0x50 seja comparado com 0x60, 0x51 com 0x61, seguindo esse passo até 0x54 e 0x64.

De acordo com a funcionalidade do **CJNE**, caso os operandos sejam iguais, ele prossegue para a próxima instrução, caso contrário ele salta para uma posição de memória definida.

Assim, se o usuário inseriu corretamente a sequência, ao passar por todas comparações, é chamada a rotina que escreve no LCD a mensagem *Winner*. Caso o algoritmo encontre um erro na sequência é chamada a rotina que escreve no LCD a mensagem *Loser!*. Após a decisão de vitória ou derrota, o usuário pode reiniciar o jogo pressionando a chave SW0, a mesma utilizada para iniciar o jogo.

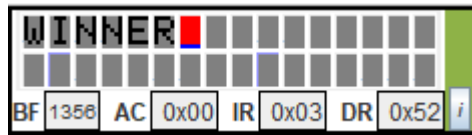


Figure 8: Mensagem de vitória após comparação entre sequências. Fonte: Ed-Sim51

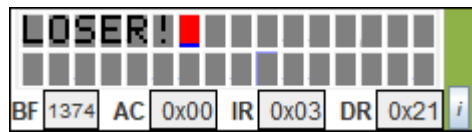


Figure 9: Mensagem de derrota após comparação entre sequências. Fonte: Ed-Sim51

4 Código fonte

```
1  ; AUTHORS: VITOR ACOSTA DA ROSA
2  ;                               : ANDY SILVA BARBOSA
3
4  ; Frequencia utilizada: 8
5  ; R7 -> Não usado.
6  ; R6 -> Marcador para DELAYS.
7  ; R5 -> Não usado.
8  ; R4 -> Não usado.
9  ; R3 -> Delay para o LCD
10 ; R2 -> Marcador para rotacionar os bits da sequência gerada pelo 8051.
11 ; R1 -> Ponteiro para memória, serve para salvar a sequência gerada pelo 8051.
12 ; R0 -> Ponteiro para memória, serve para salvar a sequência inserida pelo usuário.
13
14 ORG 0000h ;RESET
15
16 ;----- CONFIGURAÇÃO DO LCD -----
17 ; --- Mapeamento de Hardware (8051) ---
18     RS     equ     P1.3    ;Reg Select ligado em P1.3
19     EN     equ     P1.2    ;Enable ligado em P1.2
20
21 LJMP CONFIG ;PULA PARA A ROTINA CONFIG
22
23 ;-----INTERRUPÇÃO EXTERNA 0-----
24 ORG 0003h
25 ; Rotina INT_EXT0:
26 ; Através da interrupção externa 0, ativada pelo P3.2 o usuário poderá
27 ; começar o jogo. Se pressionar novamente, o jogo pausa.
28 INT_EXT0:
29     AJMP START_GAME
30     RETI
31
```

```

32 ;-----INTERRUPÇÃO DO TEMPORIZADOR 0-----
33 ORG 000Bh
34 INT_TEMP0:
35     MOV TH0, #0 ;Move para o valor de recarga do contador o valor 0.
36     MOV TL0, #0 ;Move para o contador o valor 0.
37     RETI
38
39
40 ;----- CÓDIGO PRINCIPAL -----
41 ORG 0080h
42 ;----- FUNÇÕES AUXILIARES DO JOGO -----
43 ESCREVE_LOSE:
44     acall lcd_init
45     MOV A, #0
46     ACALL posicionaCursor
47     MOV A, #'L'
48     ACALL sendCharacter
49     MOV A, #'O'
50     ACALL sendCharacter
51     MOV A, #'S'
52     ACALL sendCharacter
53     MOV A, #'E'
54     ACALL sendCharacter
55     MOV A, #'R'
56     ACALL sendCharacter
57     MOV A, #'!'
58     ACALL sendCharacter
59     ACALL retornaCursor
60     MOV P1, #11111111b
61

```

```

62      ;----- RESET DO JOGO -----
63      CPL P0.1 ;Limpa a FLAG
64      MOV R1, #80 ;Redefine o apontamento
65      MOV R0, #96 ;Redefine o apontamento
66
67      JNB P3.2, $ ;Enquanto o usuário não apertar SW0, o jogo não reinicia.
68      LJMP PRE_GAME
69
70  ESCREVE_WIN:
71      acall lcd_init
72      MOV A, #0
73      ACALL posicionaCursor
74      MOV A, #'W'
75      ACALL sendCharacter
76      MOV A, #'I'
77      ACALL sendCharacter
78      MOV A, #'N'
79      ACALL sendCharacter
80      MOV A, #'N'
81      ACALL sendCharacter
82      MOV A, #'E'
83      ACALL sendCharacter
84      MOV A, #'R'
85      ACALL sendCharacter
86      ACALL retornaCursor
87      MOV P1, #11111111b
88
89      ;----- RESET DO JOGO -----
90      CPL P0.1 ;Limpa a FLAG
91      MOV R1, #80 ;Redefine o apontamento
92      MOV R0, #96 ;Redefine o apontamento
93
94      JNB P3.2, $ ;Enquanto o usuário não apertar SW0, o jogo não reinicia.
95      LJMP PRE_GAME

```

```

98 ; ROTINA DELAY_ARMAZENAMENTO
99 ; Gera um delay de 192ms entre cada inserção do usuário
100 ; (Dado que DJNZ consome 2us por execução, e que 0x60 = 96decimal temos 2us * 96 = 192 ms)
101 DELAY_ARMAZENAMENTO:
102     DJNZ R6, DELAY_ARMAZENAMENTO
103     MOV R6,#60
104     RET
105
106 ; ROTINA ROTATE
107 ; Rotaciona a sequência de 8bits inserida no acumulador A, B vezes.
108 ROTATE:
109     RR A
110     DJNZ B, ROTATE
111     MOV P1, A
112     RET
113
114 ; ROTINA SALVA_SEQ
115 ; Rotina que percorre a memória na qual R1 aponta e salva o código
116 ; binário do acendimento dos LEDS nessa posição (em HEX)
117 SALVA_SEQ:
118     MOV @R1, P1
119     INC R1
120     RET
121
122 ;ROTINA SALVA_USR
123 ; Rotina que percorre a memória na qual R0 aponta e salva o código
124 ; binário inserido pelo usuário nos botões SW2 até SW7 (em HEX)
125 SALVA_USR:
126     MOV @R0, P2
127     INC R0
128     CALL DELAY_ARMAZENAMENTO ;Delay para visualização do LED pressionado
129     MOV P1, #11111111b ;Apaga os LEDS
130     LJMP START_GAME

```

```

132 ;ROTINA GERA_SEED
133 ; Rotina que gera um número aleatório a partir do número disposto no Timer.
134 ; Essa rotina, serve para manter a aleatoriedade entre os LEDS acendidos.
135 ; Como o RANDOM trabalha com valores fixos, essa rotina serve para quebrar
136 ; o ciclo.
137 GERA_SEED:
138     MOV A, TL0
139     MOV B, #17
140     MUL AB
141     RLC A ;Rotaciona os bits calculados para a esquerda
142           ;considerando o Carry.
143     ADD A, B
144     MOV TL0, A
145     RET
146
147 ;-----
148
149 ;----- FUNÇÕES DO JOGO -----
150 ; ROTINA RANDOM:
151 ; Para gerar números aleatórios primeiramente é movido o valor do contador para o acumulador A
152 ; em seguida, como deseja-se adquirir o módulo, é feita a divisão por 6 o resto(B) é considerado e
153 ; o quociente(A) descartado.
154 ; Dessa forma, o resto representará a quantidade de vezes que a sequência de 8bits (inicial 01111111)
155 RANDOM:
156     CALL GERA_SEED
157     MOV P1, #11111111b
158     MOV A, TL0
159     MOV B, #6h
160     DIV AB
161
162     MOV A, #01111111b
163     MOV R2,B
164
165     CJNE R2,#0h,ROTATE
166     LJMP START_GAME

```

```

169 ; ROTINA SALVA_RANDOM
170 ; Para cada led aceso da sequência, é salvo os 8bits de P1 em uma posição de memória.
171 SALVA_RANDOM:
172     CJNE R1, #84, SALVA_SEQ ;Caso a sequência não está completa, pula para uma rotina auxiliar.
173     MOV @R1, P1
174     CPL P0.0 ;Define uma FLAG, indicando que a sequência foi salva em sua totalidade,
175             ;e permitindo que o usuário digite a sua sequência.
176     RET
177
178
179 ; ROTINA ARMAZENA_USER
180 ; Rotina que verifica e salva a sequência de botões apertados pelo usuário.
181 ARMAZENA_USER:
182     MOV P1, P2 ;Mostra qual o botão o usuário apertou
183
184     CJNE R0, #100, SALVA_USR
185     MOV @R0, P2
186
187     CPL P0.0 ;Flag que autoriza a continuação do código, já que o usuário inseriu
188             ;sua sequência.
189
190     CPL P0.1 ;Flag que permite a comparação entre a sequência gerada pelo microcontrolador
191             ;com a sequencia inserida pelo usuário.
192     LJMP START_GAME

```

```

194 ; ROTINA LOOP_INSERT
195 ; Espera que o usuário pressione pelo menos um botão entre SW2 e SW7, para
196 ; comparar com a sequência do jogo.
197 LOOP_INSERT:
198     MOV P1, #01111111b
199     MOV P1, #11111111b
200     JNB P2.7, ARMAZENA_USER
201     JNB P2.6, ARMAZENA_USER
202     JNB P2.5, ARMAZENA_USER
203     JNB P2.4, ARMAZENA_USER
204     JNB P2.3, ARMAZENA_USER
205     JNB P2.2, ARMAZENA_USER
206     SJMP LOOP_INSERT
207
208 ; ROTINA COMPARA_JOGO
209 ; Rotina responsável por comparar a sequência gerada aleatoriamente com
210 ; a sequência inserida pelo usuário, a fim de mostrar a vitória ou a derrota.
211 COMPARA_JOGO:
212     MOV P1, #11111111b
213     MOV A, 80
214     CJNE A, 96, ERRO
215     MOV A, 81
216     CJNE A, 97, ERRO
217     MOV A, 82
218     CJNE A, 98, ERRO
219     MOV A, 83
220     CJNE A, 99, ERRO
221     MOV A, 84
222     CJNE A, 100, ERRO
223     LJMP ESCRIVE_WIN
224
225 ERRO:
226     LJMP ESCRIVE_LOSE

```



```

228 ;----- CONFIGURAÇÕES DO JOGO -----
229 CONFIG:
230     MOV R6, #60
231
232 ;----- APONTAMENTOS INICIAIS -----
233 ;Apontamento inicial para a posição de memória onde ficará salvo as sequencias de LEDS
234     MOV R1, #80
235     MOV R0, #96
236 ;-----
237
238 ;----- CONFIGURAÇÕES DAS INTERRUPÇÕES EXTERNAS -----
239     SETB IT0 ;Define o tipo de interrupção externa sendo
240                ;executada toda vez que ocorre uma borda de descida
241                ;no pino P3.2.
242     SETB EX0 ;Habilita a interrupção externa 0 do registrador
243     SETB IT1 ;Define o tipo de interrupção externa sendo
244                ;executada toda vez que ocorre uma borda de descida
245                ;no pino P3.3.
246     SETB EX1 ;Habilita a interrupção externa 1 do registrador
247     SETB EA ;Habilita as interrupções
248 ;-----
249
250 ;----- INTERRUPÇÕES DO TEMPORIZADOR -----
251     MOV TMOD,#2 ;Modo 2 - Temporizador/Contador de 8 bits com recarga automática.
252     MOV TH0, #0 ;Move para o valor de recarga do contador o valor 0.
253     MOV TL0, #0 ;Move para o contador o valor 0.
254     SETB ET0 ;Habilita a interrupção do contador 0.
255     SETB TR0 ;LIGA O CONTADOR 0
256 ;-----
257
258 ; ROTINA PRE_GAME
259 ; Assegura que o usuário aperte o botão para que o jogo comece
260 PRE_GAME:
261     JB P3.2, PRE_GAME

```

```

263  START_GAME:
264      JNB P0.0, LOOP_INSERT
265      JNB P0.1, COMPARA_JOGO
266      CALL RANDOM
267      CALL SALVA_RANDOM
268      SJMP START_GAME
269
270  ;----- ROTINAS PARA LCD -----
271  lcd_init:
272
273      CLR RS
274
275      CLR P1.7
276      CLR P1.6
277      SETB P1.5
278      CLR P1.4
279
280      SETB EN
281      CLR EN
282
283      CALL delay
284
285      SETB EN
286      CLR EN
287
288      SETB P1.7
289
290      SETB EN
291      CLR EN
292
293      CALL delay
294
295      CLR P1.7
296      CLR P1.6
297      CLR P1.5

```

```
298         CLR P1.4
299
300         SETB EN
301         CLR EN
302
303         SETB P1.6
304         SETB P1.5
305
306         SETB EN
307         CLR EN
308
309         CALL delay
310
311         CLR P1.7
312         CLR P1.6
313         CLR P1.5
314         CLR P1.4
315
316         SETB EN
317         CLR EN
318
319         SETB P1.7
320         SETB P1.6
321         SETB P1.5
322         SETB P1.4
323
324         SETB EN
325         CLR EN
326
327         CALL delay
328         RET
329
330
331 sendCharacter:
332         SETB RS
```

```

333      MOV C, ACC.7
334      MOV P1.7, C
335      MOV C, ACC.6
336      MOV P1.6, C
337      MOV C, ACC.5
338      MOV P1.5, C
339      MOV C, ACC.4
340      MOV P1.4, C
341
342      SETB EN
343      CLR EN
344
345      MOV C, ACC.3
346      MOV P1.7, C
347      MOV C, ACC.2
348      MOV P1.6, C
349      MOV C, ACC.1
350      MOV P1.5, C
351      MOV C, ACC.0
352      MOV P1.4, C
353
354      SETB EN
355      CLR EN
356
357      CALL delay
358      RET
359
360      ;Posiciona o cursor na linha e coluna desejada.
361      ;Escreva no Acumulador o valor de endereço da linha e coluna.
362      ;|-----|
363      ;|linha 1 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
364      ;|linha 2 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
365      ;|-----|
366      posicionaCursor:
367      CLR RS

```

```

368      SETB P1.7
369      MOV C, ACC.6
370      MOV P1.6, C
371      MOV C, ACC.5
372      MOV P1.5, C
373      MOV C, ACC.4
374      MOV P1.4, C
375
376      SETB EN
377      CLR EN
378
379      MOV C, ACC.3
380      MOV P1.7, C
381      MOV C, ACC.2
382      MOV P1.6, C
383      MOV C, ACC.1
384      MOV P1.5, C
385      MOV C, ACC.0
386      MOV P1.4, C
387
388      SETB EN
389      CLR EN
390
391      CALL delay
392      RET
393
394
395      ;Retorna o cursor para primeira posição sem limpar o display
396      retornaCursor:
397          CLR RS
398          CLR P1.7
399          CLR P1.6
400          CLR P1.5
401          CLR P1.4

```

```

403      SETB EN
404      CLR EN
405
406      CLR P1.7
407      CLR P1.6
408      SETB P1.5
409      SETB P1.4
410
411      SETB EN
412      CLR EN
413
414      CALL delay
415      RET
416
417 ;limpa o display
418 clearDisplay:
419      CLR RS
420      CLR P1.7
421      CLR P1.6
422      CLR P1.5
423      CLR P1.4
424
425      SETB EN
426      CLR EN
427
428      CLR P1.7
429      CLR P1.6
430      CLR P1.5
431      SETB P1.4
432
433      SETB EN
434      CLR EN
435
436      CALL delay
437      RET

```

```

440 delay:
441      MOV R3, #50
442      DJNZ R3, $
443      RET

```

5 Imagens da simulação realizada na IDE

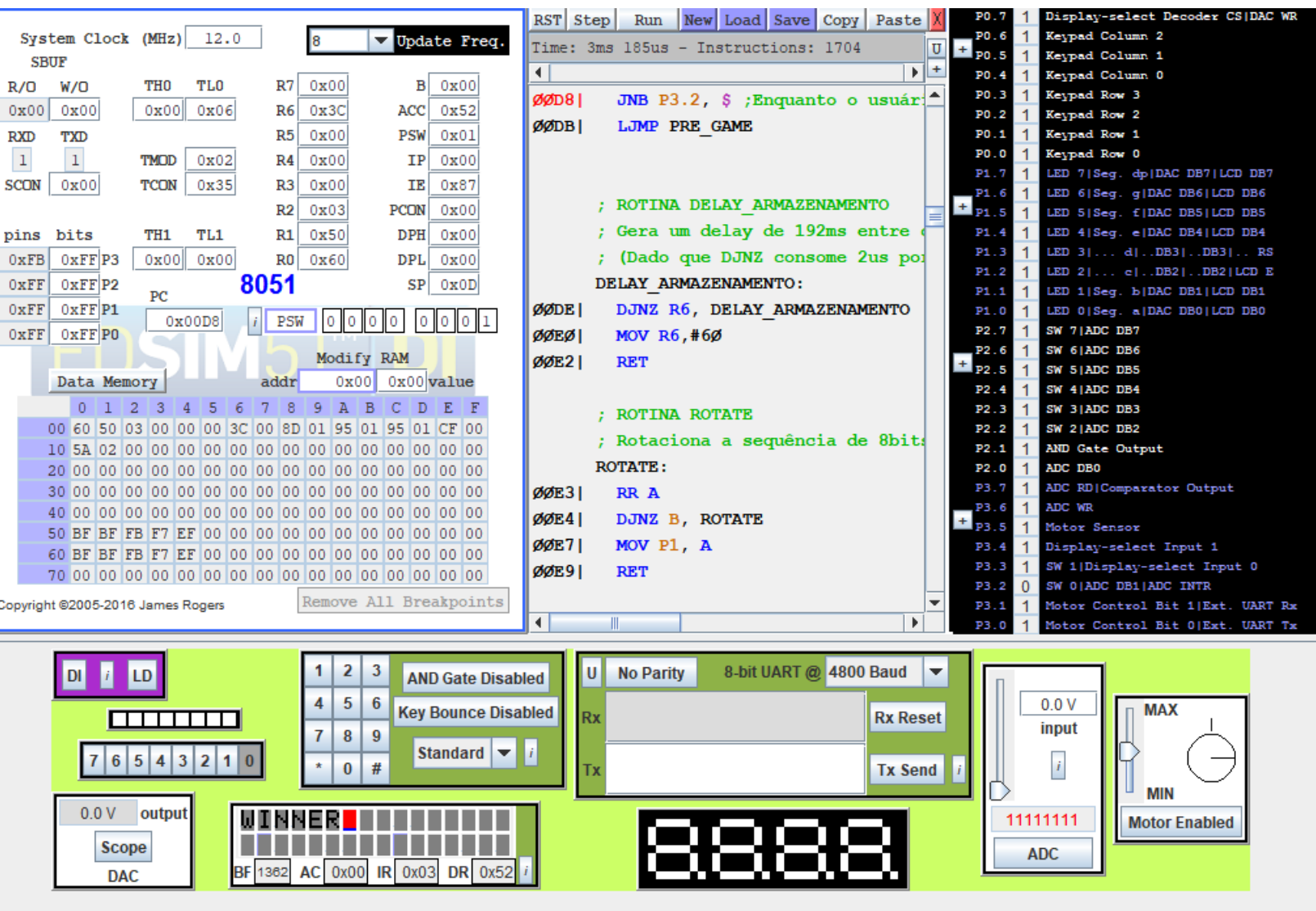


Figure 10: Condição de vitória no jogo. Fonte: EdSim51

Perceba que, o conteúdo na posição de memória 0x50 é igual à 0x60, e repete-se até 0x54 e 0x64, o que indica a vitória do usuário, e o tratamento esperado.

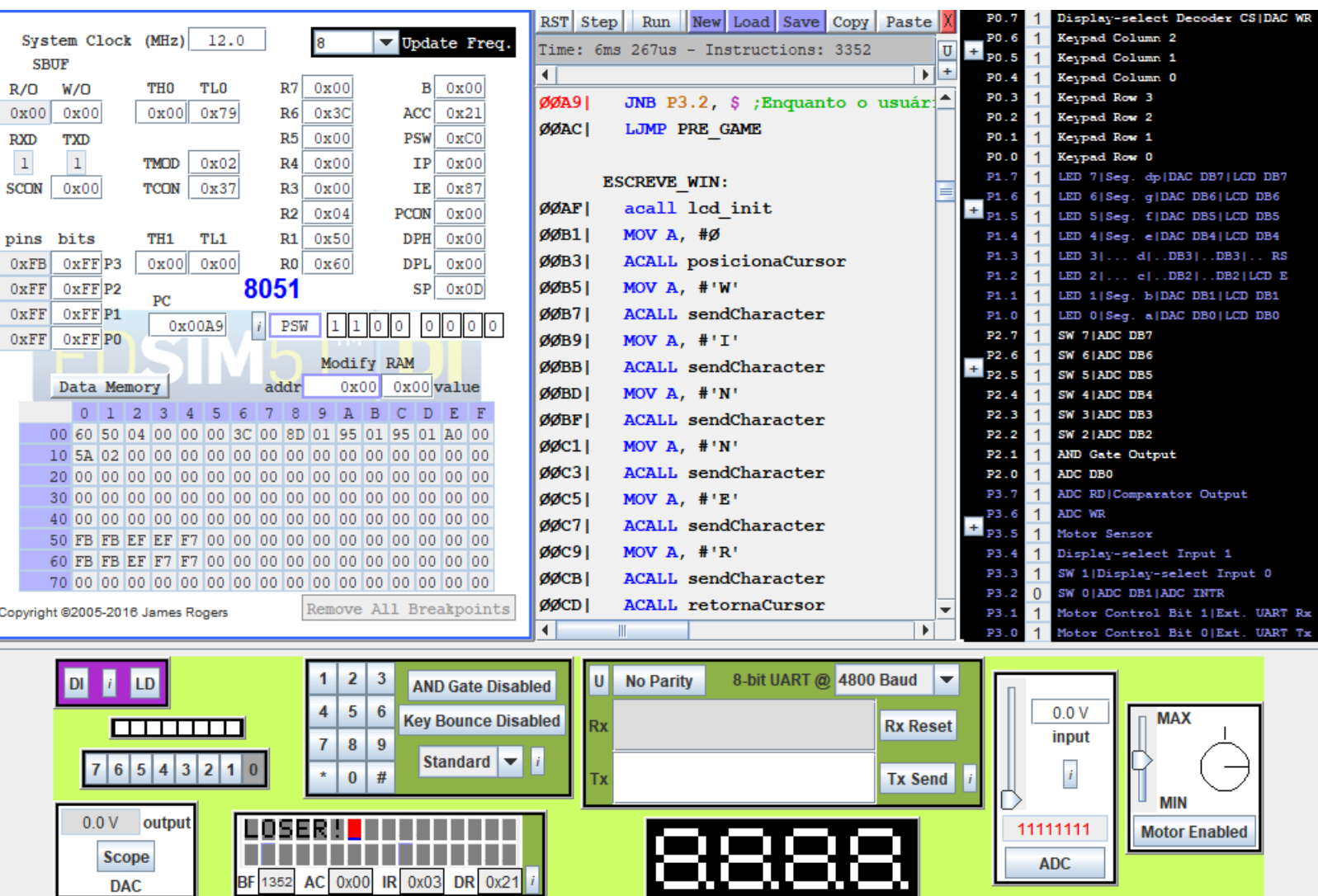


Figure 11: Condição de derrota no jogo. Fonte: EdSim51

Com excessão dos valores nas posições de memória 0x53 e 0x63, todos os outros valores coincidem, na comparação, ao primeiro erro, a mensagem *Loser!* será escrita, não havendo a necessidade de verificar as demais posições da memória. Perceba que entre a figura 10 e figura 11, o tempo de execução é quase o dobro, isso porque, ao terminar a escrita no LCD, o usuário pode recommear o jogo simplesmente apertando a chave SW0, possibilitando recommear indefinidas vezes.