


Este guia teve um caracter mais exigente embora tenhamos conseguido fazer a maior parte.

Algumas nuances podiam estar melhor, depois da conversa na aula percebemos que daria para implementar de uma melhor forma aquilo que fizemos, decidimos não alterar, porque o tempo era curto, embora para os próximos trabalhos teremos isso em consideração.

Na primeira parte deste guião era pedido para aumentar as capacidades do programa desenvolvido no guia 1, usando uart para controlar o display de 7 segmentos.

a)

A screenshot of a code editor with a dark background and light-colored text. The code is a C function named `receiveUart()`. It starts with a line number 1 and ends with 24. The function has a `char` return type. It checks if `RI0 == 1`, and if so, sets `RI0 = 0`. Then it checks if `SBUF0` is equal to 'i', 'I', or '+'. If true and `index < 15`, it increments `index`. Otherwise, it checks if `SBUF0` is equal to 'd', 'D', or '-'. If true and `index > 0`, it decrements `index`. In either case, it sets `index = SBUF0`. Finally, it returns 1. If the first condition is not met, it returns 0.

```
1 char receiveUart() {
2     if (RI0 == 1) {
3         RI0 = 0;
4
5         if (SBUF0 == 'i' || SBUF0 == 'I' || SBUF0 == '+') {
6             if (index < 15) {
7                 index++;
8             }
9         } else {
10            if (SBUF0 == 'd' || SBUF0 == 'D' || SBUF0 == '-') {
11                if (index > 0) {
12                    index--;
13                }
14            } else {
15                index = SBUF0;
16            }
17        }
18
19        return 1;
20    }
21
22    return 0;
23 }
24
```

Figura referente ao ponto i) e ii).

iii)

```
1 void sendIndex() {
2     char charToBuffer;
3
4     if (index != prev_index) {
5         while (TI0 == 0);
6
7         // Converte o valor hex para caractere ASCII
8         if (index < 10) {
9             charToBuffer = index + '0';
10        } else {
11            charToBuffer = index - 0x0A + 'A';
12        }
13
14        SBUF0 = charToBuffer;
15        TI0 = 0;
16        prev_index = index;
17    }
18
19    P2 = vetor[index] | ((char)dp_state << 7);
20 }
```

Se o index for menor que 10 ele envia o programa faz o seguinte, pega no valor de 0 na tabela ASCII e junta o valor do index ex: 0 -> 0x30,  $0x30 + 1 = 0x31$  valor da tabela ASCII para o 1.

Quando é maior que 9, o programa retira o valor de 0x0A do index e adiciona o valor de 'A' fazendo assim valores de A a F usando a logica de cima.

b)

i)

```
1 void timer2_isr(void) interrupt 5 using 2 {
2
3     char P2_state;
4     P2_state = P2;
5
6     // sysclock /12 = 4MHz
7     // 1s 4,000,000 ticks!
8     // 10ms => 40000 ticks!
9     // 50 ciclos = 500ms
10
11     blink--;
12     if(blink == 0){
13         dp_state = ~dp_state;
14         blink = 50;
15     }
16
17     seg_dp = dp_state;
18     TF2H = 0;
19 }
```

Usando o código dado pelo professor, fizemos algumas alterações, para poder contar 500ms, alterando o valor do “blink” para 50

ii)

```
1 P2 = vetor[index] | ((char)dp_state << 7);
```

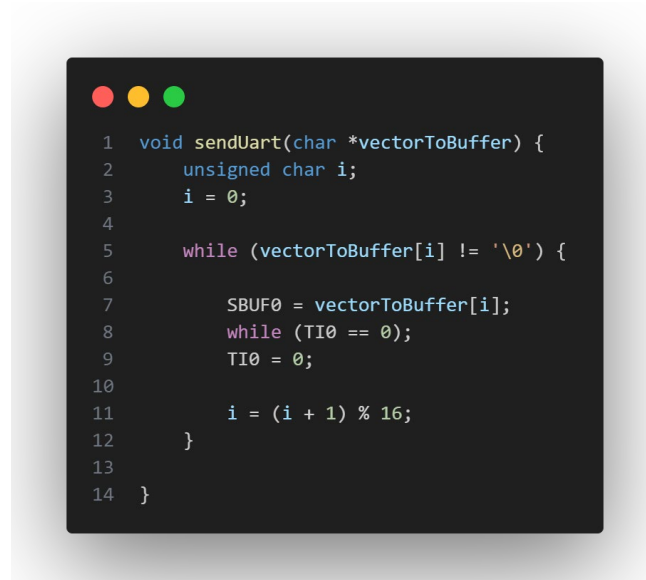
No ponto ii) usamos operações bitwise para poder preservar o valor anterior no display.

### Guião 3 Parte 2:

a)

i)

Esta foi a parte que faltou completar, a ideia está lá, faltou tempo, para perceber o que se passa.



A função recebe um vetor e vai mandando para o buffer (SBUF0) os caracteres um a um até quando chega o caracter final o '\0'

ii)



É criado um vetor “inputKeyBuffer” que gere a entrada do uart, e quando recebe a chave por defeito 8051 muda de estado para o aberto, caso contrário vai para o estado de errado, a figura em cima mostra como foi feito a deteção da chave, na conversa com o professor detetamos algumas falhas nesta implementação, o correto seria fazer como o professor disse na aula, ler do Buffer geral com condições de final (0x0a e 0x0d), assim não corríamos o risco de ler coisa fora do código metido.

iii)

```
1 char uartAlarm() {
2
3     if (RI0 == 1) {
4         RI0 = 0;
5
6         if (bufferAlarmIndex < 5) {
7             // lê o que está no buffer
8             inputAlarmBuffer[bufferAlarmIndex] = SBUF0;
9             bufferAlarmIndex++;
10        }
11
12        if (bufferAlarmIndex == 5) {
13            if (inputAlarmBuffer[0] == 'r' && inputAlarmBuffer[1] == 'e' && inputAlarmBuffer[2] == 's' && inputAlarmBuffer[3] == 'e' && inputAlarmBuffer[4] == 't'){
14                nextstate = S1;
15                bufferAlarmIndex = 0;
16                return 0;
17            }
18            else {
19                bufferAlarmIndex = 0;
20            }
21        }
22    }
23
24    return 1;
25 }
26 }
```

É usado a mesma implementação do ponto ii), embora seja igual ao ponto ii) a palavra reset tem de ser enviada caracter a caracter via uart, não sabemos porque, mas suspeitamos que seja problema do buffer

iv)

```
1 void uartAdmin() {
2
3     if (RI0 == 1) {
4         RI0 = 0;
5         if (bufferAdminIndex < 3) {
6             // le o que está no buffer
7             inputAdminBuffer[bufferAdminIndex] = SBUF0;
8             bufferAdminIndex++;
9         }
10        else {
11            bufferAdminIndex = 0;
12        }
13
14        if (bufferAdminIndex == 3) {
15            if (inputAdminBuffer[0] == '*' && inputAdminBuffer[1] == '#'){
16                if (inputAdminBuffer[2] == 'L'){
17                    nextstate = S1;
18                    bufferAdminIndex = 0;
19                }
20                if (inputAdminBuffer[2] == 'O'){
21                    nextstate = S2;
22                    bufferAdminIndex = 0;
23                }
24                if (inputAdminBuffer[2] == 'P'){
25                    nextstate = S5;
26                    bufferAdminIndex = 0;
27                }
28                if (inputAdminBuffer[2] == 'E'){
29                    nextstate = S3;
30                    bufferAdminIndex = 0;
31                }
32                if (inputAdminBuffer[2] == 'A'){
33                    nextstate = S4;
34                    bufferAdminIndex = 0;
35                }
36            }
37        }
38    }
39 }
```

Usamos a mesma implementação ao ponto ii) e iii) mas neste caso a logica é um pouco diferente como para mudar de estado é usado uma trama especial \*#<estado pretendido> no nosso caso os códigos são: \*#L para ir para o estado fechado; \*#O para ir para o estado aberto; \*#P para ir para o estado programar onde se pode mudar a chave; \*#E para ir para o estado do erro; \*#A para ir para o estado de alarme, o modo admin, como nos lhe chamamos é apenas acessível a quem sabe a chave, fizemos isto, porque os buffers entravam em conflito, devido a serem locais, se fosse o buffer global já não teríamos esse problema

b)

Em teoria a implementação é quase a correta, mas diverge em alguns pontos, na nossa implementação os buffers usados eram criados específicos para as funções, e não um Buffer geral, de onde as funções pudessem ler, como a nossa dúvida foi tirada na aula, não deu tempo para implementar como gostaríamos.

```
1 void UARTInterrupt(void) interrupt 4 {
2     if (RI0) {
3         RI0=0;
4         if (RxNumberOfData < BUFFERSIZE) {
5             RxBuffer[RxBufWritePtr]=SBUF0 ;
6             RxBufWritePtr = (RxBufWritePtr +1) % BUFFERSIZE;
7             RxNumberOfData++;
8         }
9     }
10    if (TI0) {
11        TI0=0;
12        TxNumberOfData--;
13        if (TxNumberOfData) {
14            SBUF0=TxBuffer[TxBufReadPtr];
15            TxBufReadPtr = (TxBufReadPtr +1) % BUFFERSIZE;
16            //TxNumberOfData--;
17        }
18    }
19 }
```

Devíamos ter implementado algo do género...