**Arquitetura de Computadores**

**2020/2021**



**3º Projeto**

**Sistema de controlo do número de entradas em espaços fechados**

**Docentes:**

* João Dionísio Simões Barros
* Nuno Fábio Gomes Camacho Ferreira
* Pedro Miguel Pestana Camacho
* Sofia Isabel Silva Inácio

**Membros do grupo:**

* Sérgio Oliveira nº 2049719;

**ÍNDICE**

[**Introdução** 3](#_Toc74736576)

[**Objetivo** 4](#_Toc74736577)

[**Desenvolvimento** 4](#_Toc74736578)

[**Endereçamento** 5](#_Toc74736579)

[**Conclusão** 6](#_Toc74736580)

[**Anexo A (fluxograma)** 6](#_Toc74736581)

[**Anexo B (código em linguagem assembly)** 7](#_Toc74736582)

[**Anexo C (código em linguagem C)** 7](#_Toc74736583)

**ÍNDICE DE ILUSTRAÇÕES**

[**Figura 1 -** C 4](#_Toc74736457)

[Figura 2 - Assembly 5](#_Toc74736458)

[Figura 3 - Sistema de controlo do número de pessoas em espaços fechados 7](#_Toc74736459)

# **Introdução**

O presente trabalho prático foi realizado no âmbito curricular de Arquitetura de Computadores, na qual foi proposto realizar um sistema de controlo do número de entradas em espaços fechados. O trabalho foi desenvolvido em C e assembly, e para o realizar necessitei de recorrer ao software *Kail*, e ao software *Multisim*, que me permitiram simular o código por mim criado.

De início, foi necessário perceber o funcionamento da arquitetura do processador 8051, que possui um set de instruções semelhante, mas diferente do processador PEPE, ao qual estava habituado.

Para tal, recorri ao seguinte link <https://www.keil.com/support/man/docs/is51/is51_instructions.htm>, link este que eu extremamente recomendo a qualquer programador que pretenda desenvolver um programa para este processador, e ao qual devo grande parte do meu sucesso ao cumprir este trabalho.

De modo a facilitar a interpretação das instruções do programa foi elaborado um fluxograma, que ajuda na compreensão do mesmo.

# **Objetivo**

Este trabalho prático teve como propósito criar um programa responsável pelo controlo do número de entradas em espaços fechados. O programa deveria permitir registar as várias saídas e entradas das pessoas no determinado espaço, e mostrar o número de vagas do estabelecimento num display de 7 segmentos.

O programa deverá manter acesa uma luz vermelha, sendo esta desligada durante 3 segundos sempre que uma pessoa dá a entrada no espaço. Durante estes 3 segundos, deve ser ligada uma luz verde.

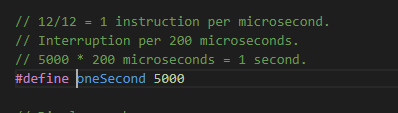
Quando apenas restam 4 ou menos vagas, dever ser ligada uma luz amarela, que pisca de 1 em 1 segundo.

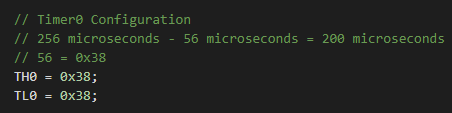
# **Desenvolvimento**

Apesar do comportamento idêntico dos dois programas criados, foi feita uma escolha de implementação que gostaria de realçar.

**C**

Na linguagem C, foi decidido que a interrupção do timer ocorresse de 200 em 200 microssegundos, após os quais uma variável é incrementada. Para saber se já passou um segundo, basta comparar esta variável com o valor 5000.





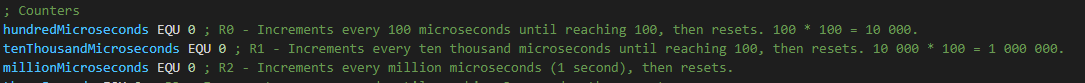


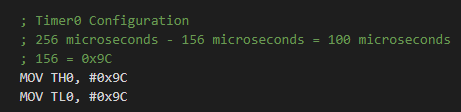
**Figura 1 -** C

**Assembly**

Na linguagem assembly, foi decidido que a interrupção do timer ocorresse de 100 em 100 *microsegundos*, após os quais a variável *hundredMicroseconds* é incrementada. Quando a variável *hundredMicroseconds* chega a 100, a variável tenThousandMicroseconds é incrementada, por sua vez, quando esta chega a 100, a variável *millionMicroseconds* é incrementada.

Esta última variável corresponde a 1 segundo. Esta escolha de implementação foi feita pois o compilador estava a truncar o valor de comparação 5000. É uma implementação que traz consigo uma diferença de desempenho negligível, visto que as variáveis intermediárias são todas elas guardadas em registos.





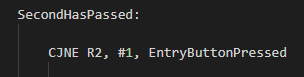


Figura 2 – Assembly

## **Endereçamento**

Todos os maiores bugs com os quais me deparei ao desenvolver este projeto foram relacionados a erros de endereçamento.

Em primeiro lugar, no endereço das interrupções coloquei o código da interrupção respetiva. Isto fez com que o código de uma interrupção fizesse *overwrite* à próxima interrupção. A solução foi usar a instrução JMP no endereço de cada interrupção para uma zona de memória segura onde a *label* da interrupção fica guardada.

Aconteceu-me exatamente o mesmo problema ao guardar as instruções do *main* a partir do endereço 0000H, e foi resolvido da mesma forma. As dores de cabeça foram imensas, e nunca mais na minha vida me vou esquecer de endereçar um programa direito.

# **Conclusão**

Na minha opinião este projeto foi bastante interessante, pois por já ter trabalhado anteriormente com o PEPE, permitiu-me ver de que forma o *instruction set* do processador no qual eu pretendo correr o programa influencia na maneira como eu resolvo o problema em assembly. As diferenças entre o PEPE e o 8051, principalmente a ausência de certos jumps condicionais neste último, fizeram com que eu abordasse o problema de maneira diferente do que abordaria no PEPE.

Eu tinha a ideia que o mapeamento entre o C e o assembly teria um resultado algo diferente, outra coisa que gostei neste projeto foi poder comparar lado a lado os dois códigos, e ver que o assembly acabou tendo uma estrutura muito parecida à do C. Tinha a ideia que o código assembly ia ficar bem mais extenso, mas no geral a maneira como resolvi o problema nas duas linguagens foi muito semelhante.

Para concluir, o programa encontra-se apresentado de uma forma clara e sem erros. A realização deste trabalho prático, permitiu-me consolidar os conhecimentos adquiridos tanto na parte prática, como na parte teórica da unidade curricular, e ampliar o meu conhecimento do 8051, e da programação nas linguagens assembly e C.

É vivamente recomendado o uso do VsCode com a extensão Amiga-Assembly para uma fácil visualização do código. Todo o código foi também formatado usado esta extensão.

# **Anexo A (fluxograma)**

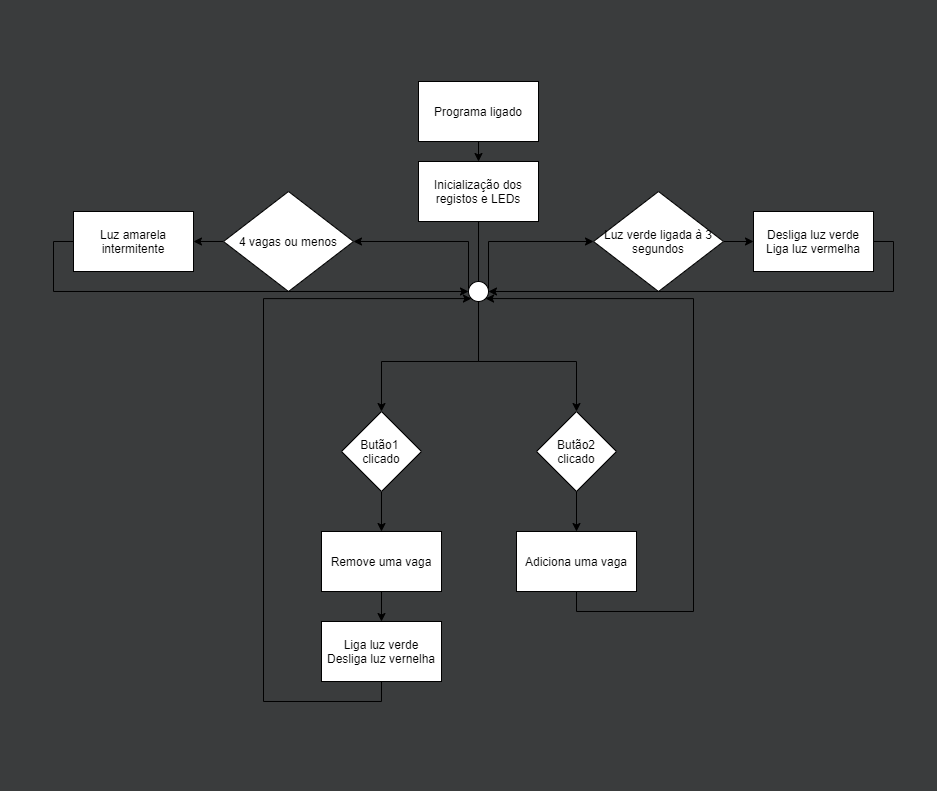


Figura 3 - Sistema de controlo do número de pessoas em espaços fechados

# **Anexo B (código em linguagem assembly)**

; Display numbers

zero EQU 0xC0

one EQU 0xF9

two EQU 0xA4

three EQU 0xB0

four EQU 0x99

five EQU 0x92

six EQU 0x82

seven EQU 0xF8

eight EQU 0x80

nine EQU 0x90

; LEDs (0 - LED Turned on)

greenLEDOff EQU P1 ^ 0

redLEDOff EQU P1 ^ 1

yellowLEDOff EQU P1 ^ 2

display EQU P2

; Counters

hundredMicroseconds EQU 0 ; R0 - Increments every 100 microseconds until reaching 100, then resets. 100 \* 100 = 10 000.

tenThousandMicroseconds EQU 0 ; R1 - Increments every ten thousand microseconds until reaching 100, then resets. 10 000 \* 100 = 1 000 000.

millionMicroseconds EQU 0 ; R2 - Increments every million microseconds (1 second), then resets.

threeSeconds EQU 0 ; R3 - Increments every second until reaching 3 seconds, then resets.

peopleLeft EQU 9 ; R4 - Amount of people that can still enter the establishment.

; Buttons

button1Pressed EQU 0 ; R5 - Button 1 state

button2Pressed EQU 0 ; R6 - Button 2 state

; Timer0Interruption

CSEG AT 000BH

JMP Timer0Interruption

; External Interruption 0 (Button 1)

CSEG AT 0003H

JMP ExternalInterruption0

; External Interruption 1 (Button 2)

CSEG AT 0013H

JMP ExternalInterruption1

CSEG AT 0000H

JMP Main

; Main

CSEG AT 1000H

Main:

CALL Initialize

For:

SecondHasPassed:

CJNE R2, #1, EntryButtonPressed

; After each second, if the green LED is on, update timerCounter2

; The green LED stays on for 3 seconds

GreenLEDon:

JB greenLEDOff, If4OrLessPeopleCanEnter ; If greenLEDOff == 0

INC R3

; Yellow LED. Turns on if there's 5 or more people inside the establishment.

; 5 or more people inside the establishment = 4 people or less left to enter the establishment.

If4OrLessPeopleCanEnter:

FourPeopleLeft:

CJNE R4, #4, ThreePeopleLeft

CPL yellowLEDOff

JMP ResetMillionMicroseconds

ThreePeopleLeft:

CJNE R4, #3, TwoPeopleLeft

CPL yellowLEDOff

JMP ResetMillionMicroseconds

TwoPeopleLeft:

CJNE R4, #2, OnePersonLeft

CPL yellowLEDOff

JMP ResetMillionMicroseconds

OnePersonLeft:

CJNE R4, #1, ZeroPeopleLeft

CPL yellowLEDOff

JMP ResetMillionMicroseconds

ZeroPeopleLeft:

CJNE R4, #0, MoreThanFourPeople

CPL yellowLEDOff

JMP ResetMillionMicroseconds

; Yellow LED. Usually off unless there's 5 or more people inside the establishment.

MoreThanFourPeople:

SETB yellowLEDOff ; Turns LED off there's 4 or less people inside the establishment

ResetMillionMicroseconds:

MOV R2, #0 ; Reset millionMicroseconds

; Entry button Pressed

EntryButtonPressed:

CJNE R5, #1, IntermedioThreeSecondsHavePassed

EntryButtonPressedCase9:

CJNE R4, #9, EntryButtonPressedCase8

CLR greenLEDOff

SETB redLEDOff

MOV display, #eight

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase8:

CJNE R4, #8, EntryButtonPressedCase7

CLR greenLEDOff

SETB redLEDOff

MOV display, #seven

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase7:

CJNE R4, #7, EntryButtonPressedCase6

CLR greenLEDOff

SETB redLEDOff

MOV display, #six

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase6:

CJNE R4, #6, EntryButtonPressedCase5

CLR greenLEDOff

SETB redLEDOff

MOV display, #five

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase5:

CJNE R4, #5, EntryButtonPressedCase4

CLR greenLEDOff

SETB redLEDOff

MOV display, #four

DEC R4

JMP EntryButtonSwitchEnd

IntermedioThreeSecondsHavePassed:

JMP ThreeSecondsHavePassed

IntermedioFor:

JMP For

EntryButtonPressedCase4:

CJNE R4, #4, EntryButtonPressedCase3

CLR greenLEDOff

SETB redLEDOff

MOV display, #three

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase3:

CJNE R4, #3, EntryButtonPressedCase2

CLR greenLEDOff

SETB redLEDOff

MOV display, #two

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase2:

CJNE R4, #2, EntryButtonPressedCase1

CLR greenLEDOff

SETB redLEDOff

MOV display, #one

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase1:

CJNE R4, #1, EntryButtonPressedCase0

CLR greenLEDOff

SETB redLEDOff

MOV display, #zero

DEC R4

JMP EntryButtonSwitchEnd

EntryButtonPressedCase0:

CJNE R4, #0, EntryButtonPressedCaseDefault

MOV display, #zero

JMP EntryButtonSwitchEnd

EntryButtonPressedCaseDefault:

JMP EntryButtonSwitchEnd

EntryButtonSwitchEnd:

MOV R5, #0 ; Resets button1 after it is pressed

; 3 seconds have passed after button 1 is pressed

ThreeSecondsHavePassed:

CJNE R3, #3, ExitButtonPressed

SETB greenLEDOff ; Turns green LED off

CLR redLEDOff ; Turns red LED on

MOV R3, #0 ; Reset threeSeconds

; Exit button pressed

ExitButtonPressed:

CJNE R6, #1, IntermedioFor

ExitButtonPressedCase0:

CJNE R4, #0, ExitButtonPressedCase1

MOV display, #one

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase1:

CJNE R4, #1, ExitButtonPressedCase2

MOV display, #two

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase2:

CJNE R4, #2, ExitButtonPressedCase3

MOV display, #three

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase3:

CJNE R4, #3, ExitButtonPressedCase4

MOV display, #four

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase4:

CJNE R4, #4, ExitButtonPressedCase5

MOV display, #five

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase5:

CJNE R4, #5, ExitButtonPressedCase6

MOV display, #six

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase6:

CJNE R4, #6, ExitButtonPressedCase7

MOV display, #seven

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase7:

CJNE R4, #7, ExitButtonPressedCase8

MOV display, #eight

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCase8:

CJNE R4, #8, ExitButtonPressedCaseDefault

MOV display, #nine

INC R4

JMP ExitButtonSwitchEnd

ExitButtonPressedCaseDefault:

JMP ExitButtonSwitchEnd

ExitButtonSwitchEnd:

MOV R6, #0 ; Resets button1 after it is pressed

JMP IntermedioFor

; External Interruption 0 (Button 1)

ExternalInterruption0:

MOV R5, #1

RETI

; External Interruption 1 (Button 2)

ExternalInterruption1:

MOV R6, #1

RETI

; Timer0 Interruption

Timer0Interruption:

INC R0 ; Increments hundredMicroseconds every 100 microseconds.

; If hundredMicroseconds hasn't reached 100, exit interruption

CJNE R0, #100, Timer0InterruptionEnd

; If hundredMicroseconds has reached 100, reset it, and increment tenThousandMicroseconds

MOV R0, #0

INC R1 ; Increment tenThousandMicroseconds

; If tenThousandMicroseconds hasn't reached 100, exit interruption

CJNE R1, #100, Timer0InterruptionEnd

; If tenThousandMicroseconds has reached 100, reset it, and increment millionMicroseconds

MOV R1, #0

INC R2 ; Increment millionMicroseconds

Timer0InterruptionEnd:

RETI

; Initialize

Initialize:

MOV R0, #hundredMicroseconds

MOV R1, #tenThousandMicroseconds

MOV R2, #millionMicroseconds

MOV R3, #threeSeconds

MOV R4, #peopleLeft

MOV R5, #button1Pressed

MOV R6, #button2Pressed

; IE Register

MOV IE, #10000111b

; TMOD Register

MOV TMOD, #00000010b

; Timer0 Configuration

; 256 microseconds - 156 microseconds = 100 microseconds

; 156 = 0x9C

MOV TH0, #0x9C

MOV TL0, #0x9C

; TCON Register

SETB IT0 ; Interruptions activate in falling edge

SETB IT1

; Initialize peripherals

SETB greenLEDOff ; Turns green LED off

CLR redLEDOff ; Turns red LED on

CLR yellowLEDOff ; Turns yellow LED on

MOV display, #nine ; Shows the number 9 on the display

; Timer0 start

SETB TR0 ; Starts the timer0

RET

END

# **Anexo C (código em linguagem C)**

#include <reg51.h>

// 12/12 = 1 instruction per microsecond.

// Interruption per 200 microseconds.

// 5000 \* 200 microseconds = 1 second.

#define oneSecond 5000

// Display numbers

#define zero 0xC0

#define one 0xF9

#define two 0xA4

#define three 0xB0

#define four 0x99

#define five 0x92

#define six 0x82

#define seven 0xF8

#define eight 0x80

#define nine 0x90

// LEDs (0 - LED Turned on)

sbit greenLEDOff = P1 ^ 0;

sbit redLEDOff = P1 ^ 1;

sbit yellowLEDOff = P1 ^ 2;

// Counters

unsigned int timerCounter1 = 0; // Increments every 200 microseconds. 5000 = 1 second has passed. Resets after a second. Yellow LED blinks every second.

unsigned int timerCounter2 = 0; // Increments every second.

unsigned int peopleLeft = 9; // Amount of people that can still enter the establishment.

// Buttons

bit button1Pressed = 0; // Button 1 state

bit button2Pressed = 0; // Button 2 state

void Timer0Interruption(void) interrupt 1 {

timerCounter1++; // Increments every 200 microseconds.

}

// External interruptions (Buttons)

void Button1Interruption (void) interrupt 0 {

button1Pressed = 1;

}

void Button2Interruption(void) interrupt 2 {

button2Pressed = 1;

}

void Initialize(void) {

// IE Register

EA = 1; // Allows all interruptions

ET0 = 1; // Activates timer0 interruptions

EX0 = 1; // Activates external interruption 0

EX1 = 1; // Activates external interruption 1

// TMOD Register

TMOD &= 0xF0; // Cleans the 4 bits from timer0

TMOD |= 0x02; // Activates mode 2 from timer0 (8 bits - autoreload)

// Timer0 Configuration

// 256 microseconds - 56 microseconds = 200 microseconds

// 56 = 0x38

TH0 = 0x38;

TL0 = 0x38;

// TCON Register

IT0 = 1; // Interruptions activate in falling edge

IT1 = 1;

// Initialize peripherals

greenLEDOff = 1; // Turns green LED off

redLEDOff = 0; // Turns red LED on

yellowLEDOff = 0; // Turns yellow LED on

P2 = nine; // Shows the number 9 on the display

// Timer0 start

TR0 = 1; // Starts the timer0

}

void main(void) {

Initialize();

for (;;) {

if (timerCounter1 >= oneSecond) {

// After each second, if the green LED is on, update timerCounter2

// The green LED stays on for 3 seconds

if (greenLEDOff == 0) {

timerCounter2++;

}

// Yellow LED. Turns on if there's 5 or more people inside the establishment.

// 5 or more people inside the establishment = 4 people or less left to enter the establishment.

if (peopleLeft <= 4) {

yellowLEDOff = !yellowLEDOff; // Alternates the yellow LED each second if there's more than 5 people inside the establishment.

}

// Yellow LED. Usually off unless there's 5 or more people inside the establishment.

else {

yellowLEDOff = 1; // Turns LED off there's 4 or less people inside the establishment

}

timerCounter1 = 0; // Reset timerCounter1

}

// Entry button pressed

if (button1Pressed == 1) {

switch (peopleLeft) {

case (9):

greenLEDOff = 0;

redLEDOff = 1;

P2 = eight;

peopleLeft--;

break;

case (8):

greenLEDOff = 0;

redLEDOff = 1;

P2 = seven;

peopleLeft--;

break;

case (7):

greenLEDOff = 0;

redLEDOff = 1;

P2 = six;

peopleLeft--;

break;

case (6):

greenLEDOff = 0;

redLEDOff = 1;

P2 = five;

peopleLeft--;

break;

case (5):

greenLEDOff = 0;

redLEDOff = 1;

P2 = four;

peopleLeft--;

break;

case (4):

greenLEDOff = 0;

redLEDOff = 1;

P2 = three;

peopleLeft--;

break;

case (3):

greenLEDOff = 0;

redLEDOff = 1;

P2 = two;

peopleLeft--;

break;

case (2):

greenLEDOff = 0;

redLEDOff = 1;

P2 = one;

peopleLeft--;

break;

case (1):

greenLEDOff = 0;

redLEDOff = 1;

P2 = zero;

peopleLeft--;

break;

case (0):

P2 = zero;

break;

default:

break;

}

button1Pressed = 0; // Resets button1 after it is pressed

}

// 3 seconds have passed after button1 is pressed

if (timerCounter2 >= 3) {

greenLEDOff = 1; // Turns green LED off

redLEDOff = 0; // Turns red LED on

timerCounter2 = 0; // Reset timerCounter2

}

// Exit button pressed

if (button2Pressed == 1) {

switch (peopleLeft) {

case (0) :

P2 = one;

peopleLeft++;

break;

case (1):

P2 = two;

peopleLeft++;

break;

case (2):

P2 = three;

peopleLeft++;

break;

case (3):

P2 = four;

peopleLeft++;

break;

case (4):

P2 = five;

peopleLeft++;

break;

case (5):

P2 = six;

peopleLeft++;

break;

case (6):

P2 = seven;

peopleLeft++;

break;

case (7):

P2 = eight;

peopleLeft++;

break;

case (8):

P2 = nine;

peopleLeft++;

break;

default:

break;

}

button2Pressed = 0; // Resets button2 after it is pressed.

}

}

}