**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

### Licenciatura em Engenharia Informática e de Computadores

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Relatório do 4.º Trabalho Prático de

Arquitetura de Computadores

**Controlador de semáforos**

### Trabalho realizado por:

### Daniel Cabaça Nº 48070

### Nuno Venâncio Nº 45824

Docente: Jorge Fonseca

### 4 de junho de 2024

**Índice**

[Descrição do Projeto 4](#_TOC_250000)

[Análise e Projeto 5](#_bookmark0)

[Especificações do jogo 5](#_bookmark1)

[Fluxograma 13](#_bookmark2)

[Circuito implementado 15](#_bookmark3)

[Rotina de interrupção 17](#_bookmark4)

[Conclusão 19](#_bookmark5)

Anexos 20

# Descrição do Projeto

**Introdução**

Este trabalho tem como principal objetivo a exploração do hardware envolvente de um processador no desenvolvimento de programas escritos em linguagem assembly. Estão envolvidos os seguintes tópicos:

* entrada e saída de dados
* temporização
* interrupções externas
* organização de programas em rotinas
* implementação de máquinas de estados em software

**Descrição do trabalho a realizar**

Pretende-se o desenvolvimento do protótipo de um sistema embebido baseado no processador

P16 que implemente o controlador de um sistema de semáforos para uma passadeira. Este sistema é composto por:

* um sinal luminoso circular para veículos, com uma única luz que pode acender com as cores vermelho e amarelo
* um sinal luminoso circular para peões, também com uma única luz, mas que pode acender com as cores vermelho e verde
* um botão de pressão para os peões solicitarem o atravessamento da faixa de rodagem. O sistema inclui ainda quatro interruptores para a configuração do tempo que o sinal verde de travessia de peões deve estar aberto

**Arquitetura do protótipo**

A diagram of a computer

Description automatically generated

**Figura 1**: Diagrama de blocos do protótipo do sistema

**Especificação do funcionamento do sistema**

* Normalmente, o sistema encontra-se num estado em que o LED L1 apresenta uma luz amarela a piscar ao ritmo de 0,5 segundos (amarelo intermitente) e o LED L2 apresenta uma luz vermelha permanentemente acesa
* A deteção de uma transição ascendente ('0' → '1') no sinal PEDESTRIAN\_BUTTON faz evoluir o sistema para um estado em que o LED L1 apresenta uma luz vermelha permanentemente acesa e o LED L2 apresenta uma luz verde permanentemente acesa
* O sistema mantém-se neste estado, no mínimo, durante um período CROSSING\_TIME. Este tempo deve ser estendido por iguais períodos sempre que for detetada uma nova transição ascendente no sinal PEDESTRIAN\_BUTTON
* O sistema retorna ao estado original logo que o tempo de espera se esgote
* Independentemente do estado em que o sistema se encontre, o LED L3 deverá apresentar uma luz verde permanentemente acesa
* A qualquer momento o sistema pode alternar para o modo de configuração, bastando para tal que o sinal CONFIG passe a tomar o valor '1'

No modo configuração, o sistema deve cumprir o seguinte funcionamento:

* O LED L3 deverá apresentar uma luz amarela permanentemente acesa, enquanto os LED L1 e L2 devem apresentar, respetivamente, luzes amarela e verde a piscarem ao ritmo de 0,5 segundos
* O utilizador pode definir o valor do período CROSSING\_TIME alterando a combinação nos bits da entrada TIME. O sistema tem um valor pré-definido de 10 segundos
* A qualquer momento o sistema pode retornar ao modo de operação, bastando para tal que a entrada CONFIG volte a tomar o valor '0'

# Solução da ligação SDP16 – pTC

De modo a estabelecer a comunicação entre o processador e o timer, será necessário fazer ligações entre os dois componentes.

A group of black and white text

Description automatically generated

**Figura 2:** Ligações P16

A diagram of a circuit board

Description automatically generated

**Figura 3:** Ligações pTC

Para o objetivo pretendido, o seguinte esquema mapeia as ligações entre ambos componentes

A diagram of a circuit board

Description automatically generated

**Figura 4:** Ligação P16 - pTC

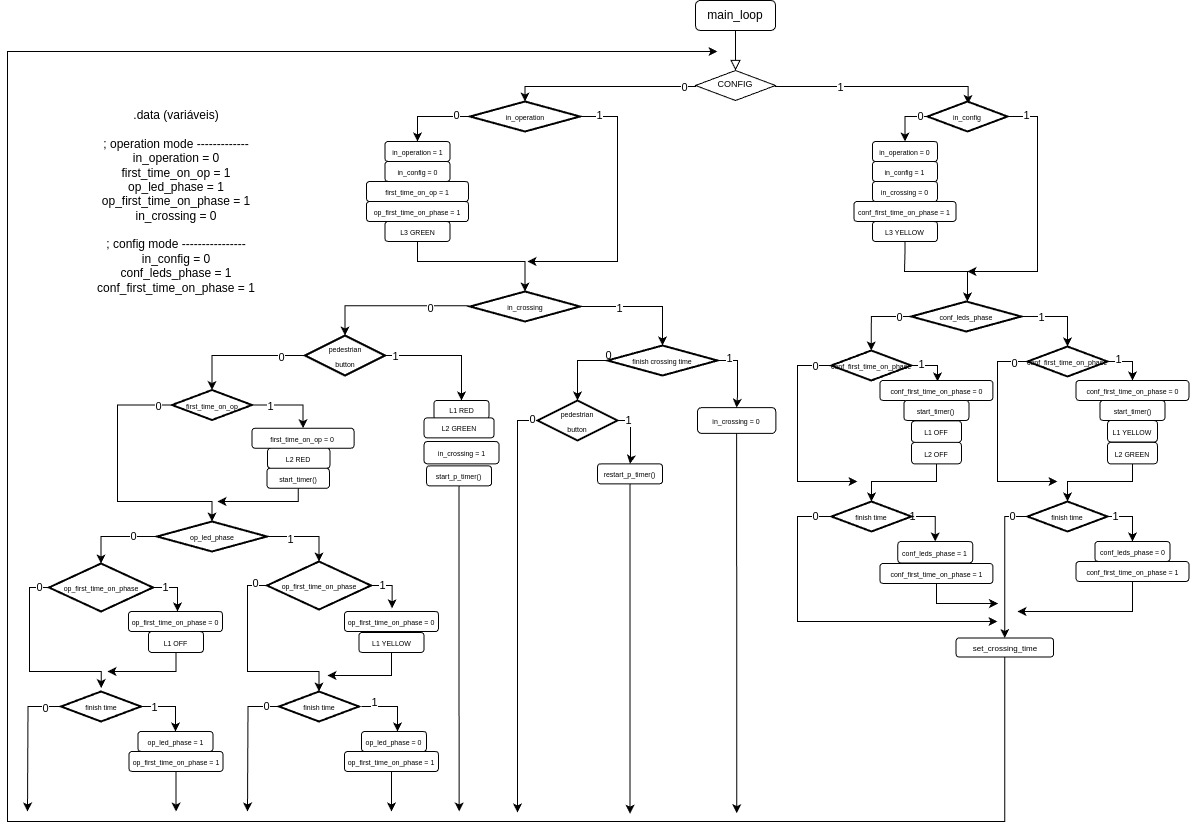
A table with numbers and letters

Description automatically generated

**Figura 5:** Mapeamento Ligações pTC – P16

# Flowchart

O seguinte Flowchart foi desenvolvido com o objetivo de facilitar a conceção do funcionamento global do protótipo. Adotámos uma abordagem de máquina de estado, que utiliza estados condicionados aos inputs e gerando outputs.



**Figura 6:** Flowchart da solução

**Flags**

**in\_operation:**  indica se estamos em modo operação

**first\_time\_on\_op:** indica se estamos pela primeira vez no modo de operação

**on\_led\_phase:** indica se a fase será para acender ou apagar os leds

**op\_first\_time\_on\_phase:** indica se é a primeira vez na fase

**in\_crossing:** indica se o tempo de travessia está a decorrer

**in\_config**: indica se estamos no modo de configuração

**conf\_leds\_phase**: indica se a fase será par acender ou apagar os leds

**conf\_first\_time\_on\_phase**: indica se é a primeira vez na fase

Na escolha das flag, escolhemos usar flags que indicam se estamos pela primeira vez numa fase com o objetivo de poupar clocks, uma vez que guardamos esta flag em memória uma única vez e já não necessitamos de acender/desligar os leds sempre que entramos na mesma fase que a anterior.

**Modo de operação**

Começando no main loop, a primeira verificação que fazemos é ler o bit de modo (operação – configuração) para decidir para que ramo vamos.

**Ramo de Configuração**

A primeira vez que entramos neste ramo executamos as seguintes ações

* in\_config = 1
* in\_operation = 0
* in\_crossing = 0
* conf\_first\_time\_on\_phase = 1
* colocar L3 a amarelo

Na segunda vez que passamos neste estado, vamos de imediato, sem nenhuma validação, para o próximo estado.

Os próximos estados destinam-se à operação dos leds L1 e L2, para sabermos se vamos entrar na fase de acender ou desligar, olhamos par a flag conf\_first\_time\_on\_phase, se visitarmos estas fases pela primeira vez executamos as seguintes ações:

* conf\_first\_time\_on\_phase = 0
* Inicializar o contador
* L1 OFF/YELLOW (depende da fase)
* L2 OFF/GREEN (depende da fase)

Após estas ações, verificamos se o temporizador, previamente inicializado, já chegou ao fim, se não o tiver alcançado, voltamos ao main loop e percorremos o Flowchart até chegar ao mesmo estado. Caso o tempo tenha, de facto, acabado, tomamos as seguintes ações:

* Conf\_leds\_phase = 0/1 (se for 0 passa a 1, se for 1 passa a 0)
* Conf\_first\_time\_on\_phase = 1

Sempre que saímos desta última fase, quer pelo fim do tempo, ou não, damos set no crossing\_time de acordo com o valor presente nos últimos três bits do inport.

**Ramo de Operação**

A primeira vez que entramos neste ramo executamos as seguintes ações

* in\_operation = 1
* in\_config = 0
* first\_time\_on\_op = 1
* op\_first\_time\_on\_phase = 1
* Ligar o L3 com a cor verde

Na segunda vez que passamos neste estado, vamos de imediato, sem nenhuma validação, para o próximo estado.

De seguida, verificamos se o tempo de travessia está a decorrer (in\_crossing), se for o caso, vamos ver se o tempo já terminou, se tiver terminado, colocamos in\_crossing a 0 e voltamos para o main loop. Se, por outro lado, ainda não tiver terminado, verificamos se houve alguém a carregar no botão, se sim, reiniciamos o timer, se não, não fazemos nada e saímos para o main loop.

Caso não haja ninguém a atravessar (in\_crossing), verificamos de novo o botão de pedestres, se tivermos uma mudança ascendente de sinal tomamos as seguintes ações

* Ligar L1 com a cor vermelho
* Ligar L2 com a cor verde
* In\_crossing = 1
* Inicializar o contador

Caso contrário, verificamos se é a primeira vez no modo operação, se for tomamos as seguintes ações

* First\_time\_on\_op = 0
* Ligar L2 com cor vermelho
* Inicializar o contador

Se não for a primeira vez, seguimos para o próximo estado sem qualquer validação.

O próximo estado, mediante a flag op\_led\_phase é destinado a ligar/desligar o led L1. Se for a primeira vez neste estado executamos as seguintes operações

* L1 OFF/YELLOW (depende da fase)
* Op\_first\_time\_on\_phase = 0

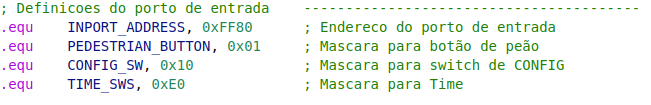
Na segunda vez, seguimos para o próximo estado sem validações prévias.

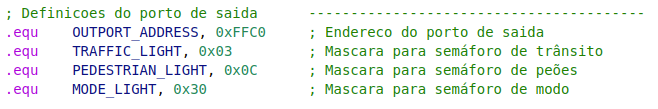
Por fim, verificamos se o tempo já terminou, se não tiver acabado, voltamos par ao main loop sem fazer nada, caso contrário as seguintes ações são executadas antes de sair para o main loop

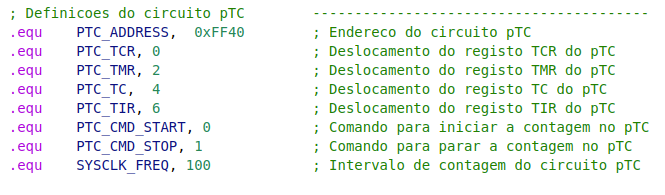
* Op\_led\_phase = 0/1 (se for 0 passa a 1, se for 1 passa a 0)
* Op\_first\_time\_on\_phase = 1

# Código Assembly

**Definições**

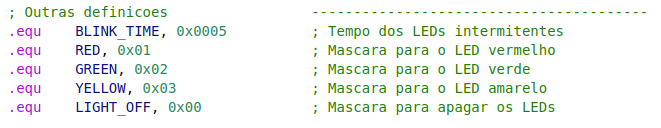
**Inport**

**Outport**

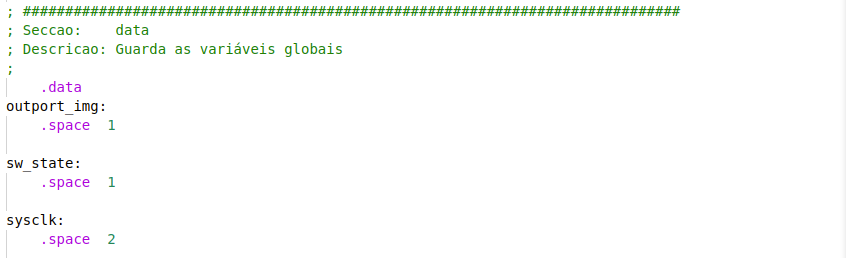
**pTC**

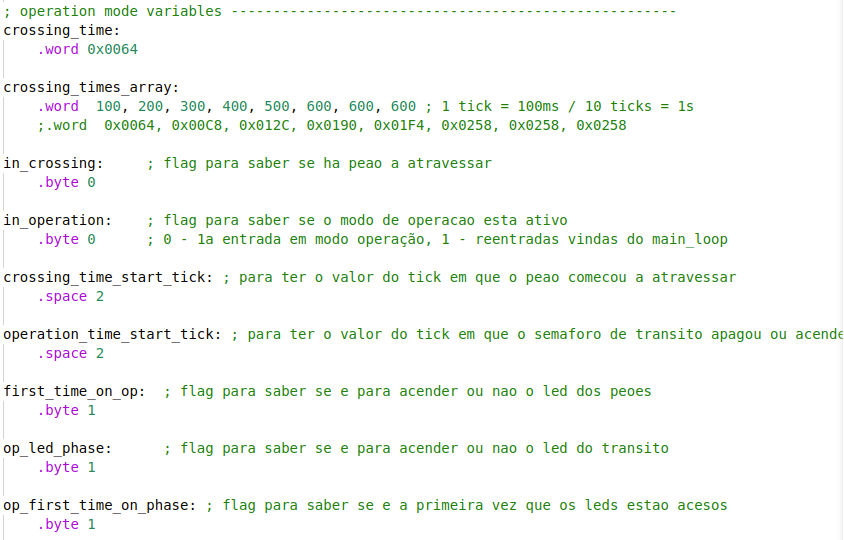
**Cálculos relativos à temporização**

Para calcular as temporizações e como precisamos de 500ms para a intermitência dos semáforos e múltiplos de 10 segundos até ao máximo de 60 para o tempo da travessia do peão, decidimos configurar cada tick a 100ms. Para tal temos de ligar o CLK com 1KHz e configurar o TMR (Timer Match Register) do pTC com o valor 99/0x0063 (0-99 = 100 ciclos) ficando cada tick a uma frequência de 10Hz = 100ms.

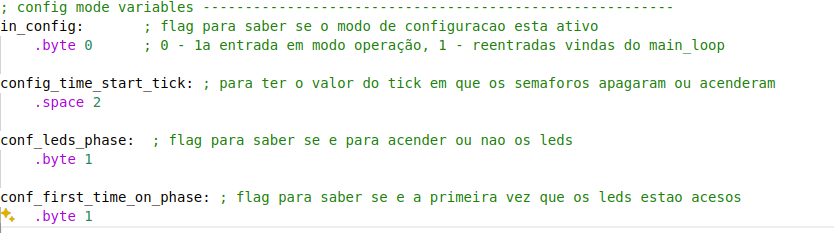
**Outras definições usadas**

**Memória**

**Variáveis globais sem valor inicial para conter o último valor escrito no output port, o estado do switch do peão e o sysclock**



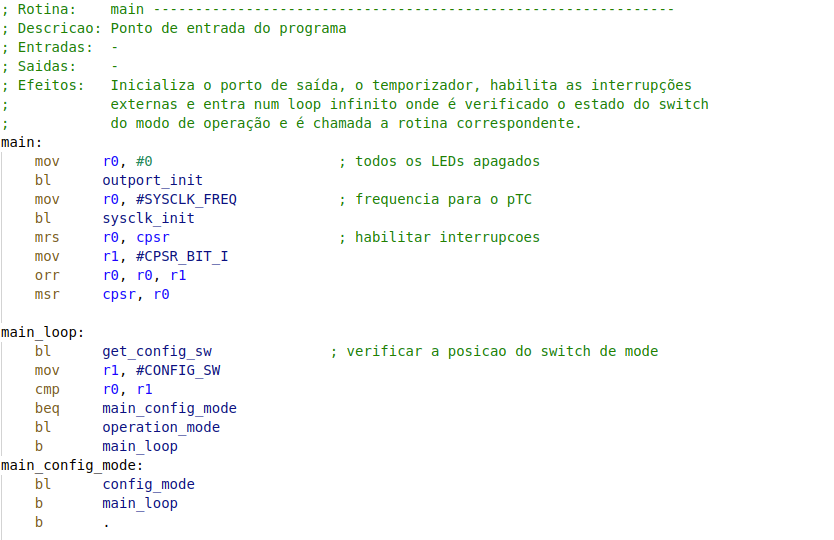
**Variáveis do modo operação guardadas em memória**



**Variáveis do modo configuração guardadas em memória**

**Execução**

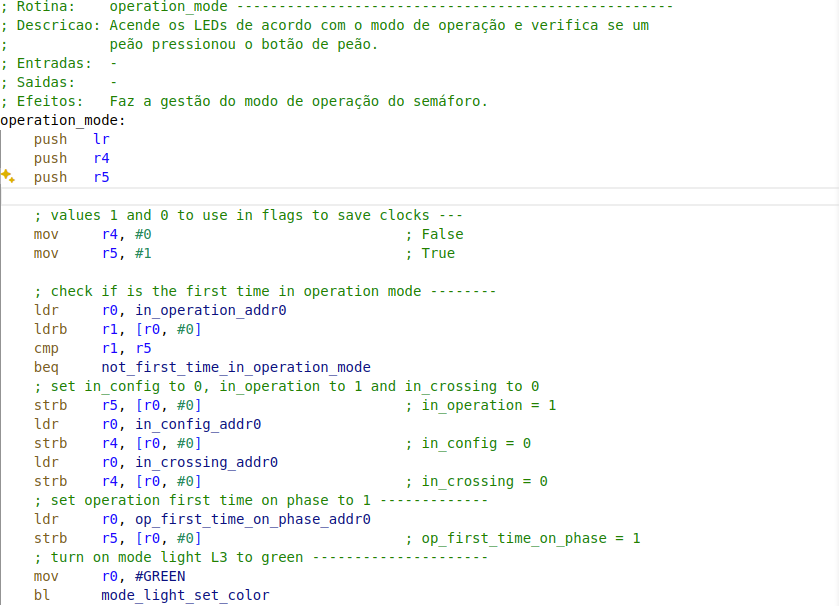
**Main**



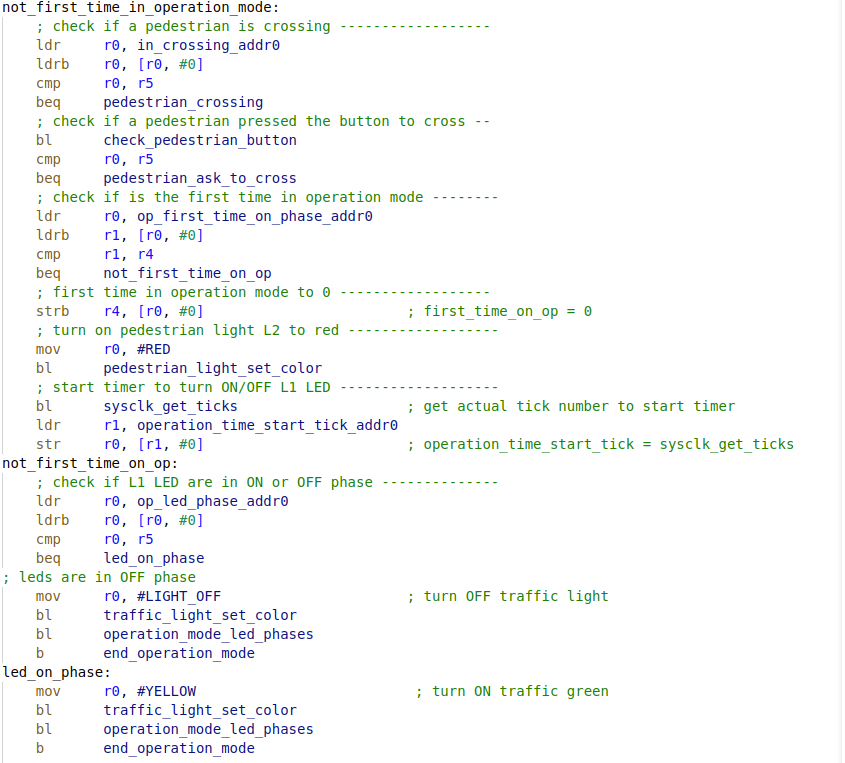
O ponto de entrada da nossa aplicação é a rotina *main* cuja funcionalidade é fazer os *inits* e configurações necessárias para o funcionamento correto do programa. Nomeadamente o *outport*, o *pTC* e habilitação do atendimento a interrupções externas.

Após a parte do *main* ter corrido, entra em *main\_loop* (estado inicial do Flowchart). Esta, é um ciclo infinito que é constantemente visitado para verificar o *inport*, percebendo se estamos em modo operação ou configuração através do valor devolvido pela rotina get\_*config*\_sw.

**Modo operação**

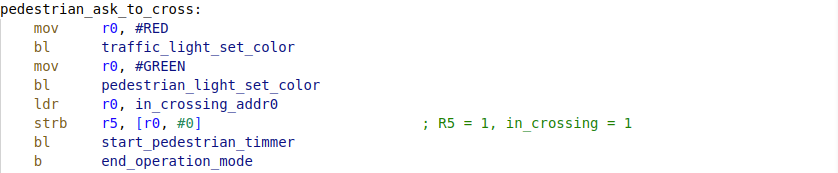


O estado inicial da rotina operação é o *operation*\_*mode* que lê o valor em *in\_operation\_addr0* para perceber se é a primeira vez em modo operação, se for a primeira vez em modo operação, os valores in\_*config* e in\_*operation* e op\_*first*\_time\_on\_phase são configurados de acordo com o estado de atuação. O LED 3 é também ligado com o valor GREEN (presente em r0) e chamando a rotina *mode\_light\_set\_color*.

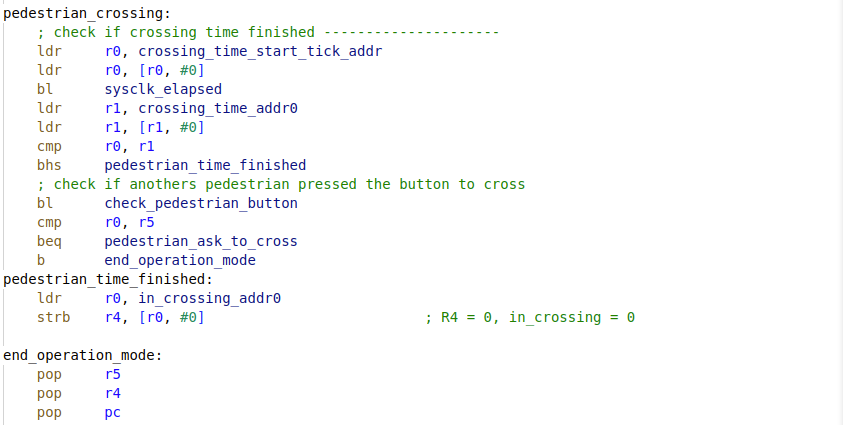


O próximo troço de código, (*not\_first\_time\_in\_operation\_mode*), começa por verificar se existe alguém em travessia através da flag in\_crossing, se não estiver ninguém, vai também verificar se houve uma transação de 1 para 0 (retorno da rotina *check\_pedestrian\_button*). Depois coloca a *flag* *first\_time\_on\_op* a zero para indicar que a próxima vez que cá entrar, já não será a primeira vez, liga o LED L2 a vermelho atravéz da rotina *pedestrian\_light\_set\_color* e vai ler e guardar o valor dos *ticks* atuais.

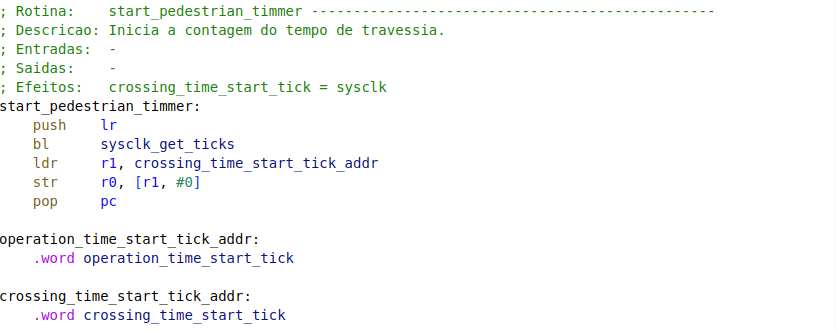
No troço *not\_first\_time\_on\_op* verificamos se é para ligar ou desligar o led através do valor armazenado na *flag op\_led\_phase* e chamando em seguida a rotina *operation\_mode\_led\_phases.*

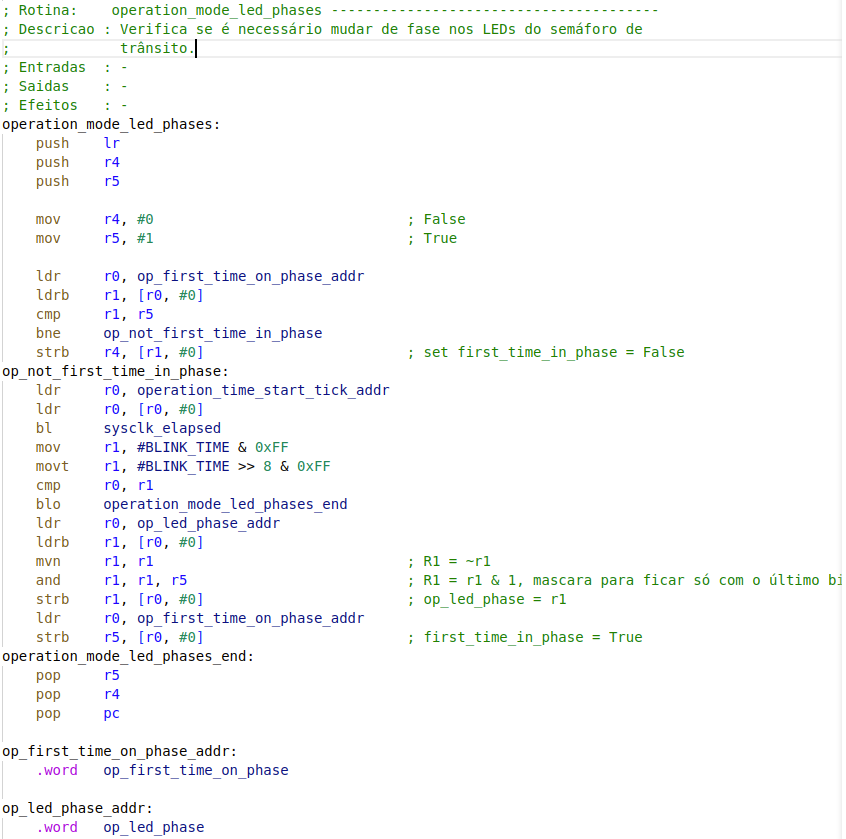


Se durante o modo operação um pedestre premir o botãodo semáforo é efetuado um salto para o troço *pedestrian\_ask\_to\_cross, onde* é ligado L1 a vermelho e L2 a verde. Colocamos o valor da *flag in\_crossing* a 1, sinalizando que está alguém na travessia da passadeira e inicializamos o respetivo timer.



Durante do troço *pedestrian\_crossing* verificamos o valor dos *ticks* em memória (o primeiro valor lido quando o peão iniciou o atravessamento) e os atuais afim de perceber se o tempo já terminou. Também importante salientar que aproveitamos para verificar novamente se durante o tempo em que o semáforo está verde para os peões alguém carregou novamente no botão, extendendo assim o tempo de atividade.

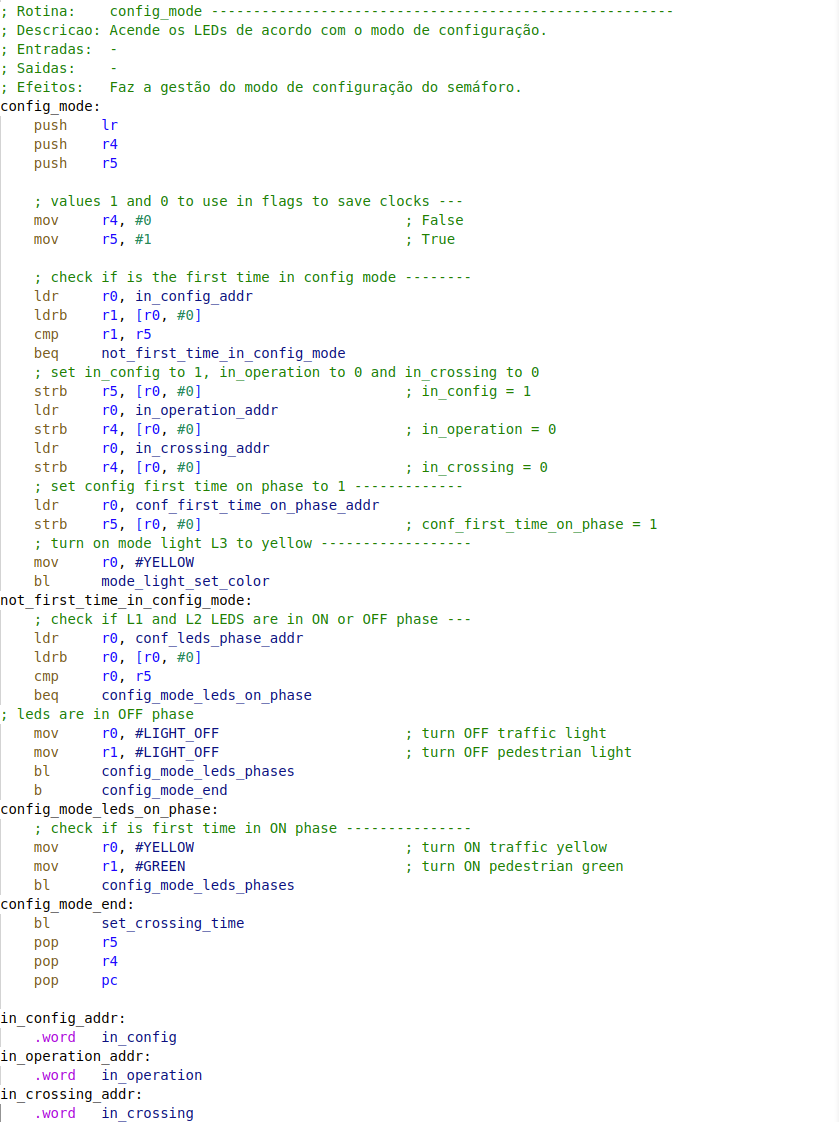


Como já vimos, sempre que alguém carrega no botão da passadeira temos de iniciar o *timer*. Simplesmente vamos ler e guardar o número de ticks atuais. 

A rotina *operation\_mode\_led\_phases* verifica se estamos nesta fase pela primeira vez e coloca a 0 a *flag op\_first\_time\_on\_phase* caso seja verdade, evitando assim repetir a escrita no *outport* para acender o LED se este já estiver aceso ou apagar se este já estiver apagado.

Em seguida, a *label* *op\_not\_first\_time\_in\_phase* é chamada quando sabemos que não é a primeira vez seguida que entramos neste estado. Vamos cálcular quantos *ticks* de diferença entre o tempo atual e o valor armazenado e comparamos, para perceber se o tempo já chegou ao fim ou não, comparando com o BLINK\_TIME. No cenário no qual o tempo terminou, invertemos o valor da flag op\_led\_phase (1 luz amarela acesa, 0 luz apagada) e recolocamos novamente a *flag op\_first\_time\_on\_phase* a 1.

**Modo configuração**

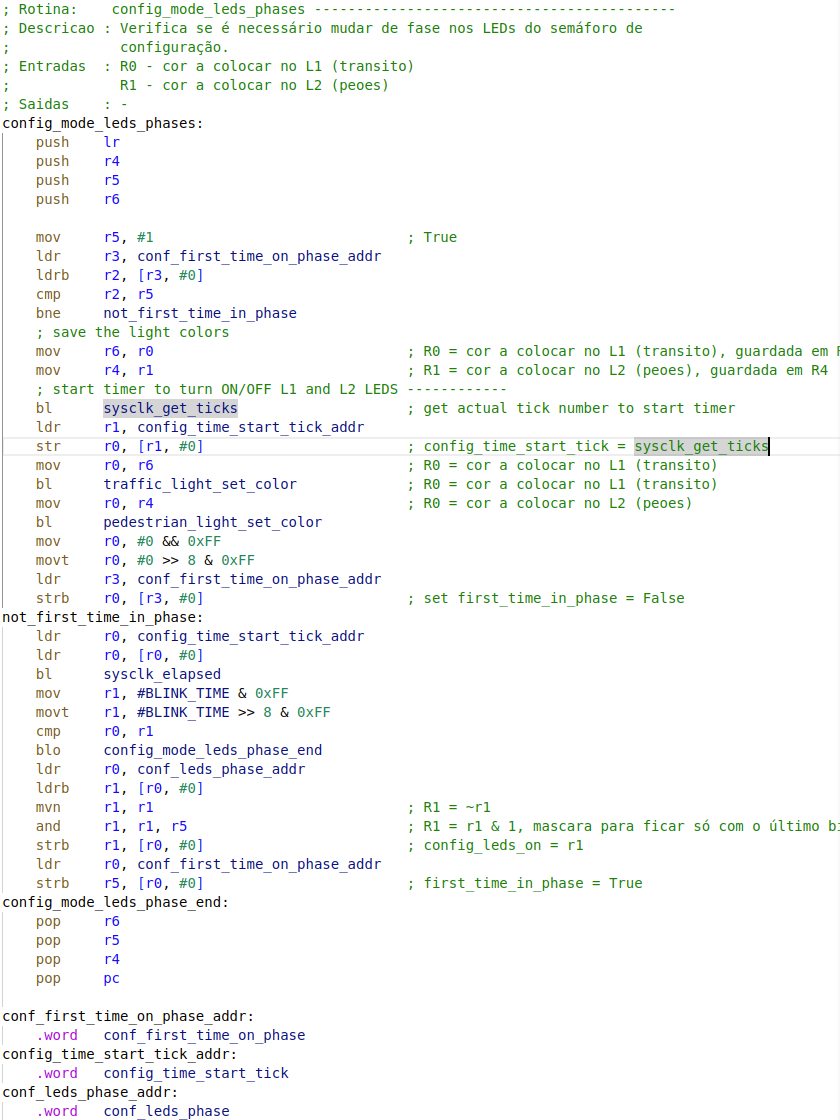


Muita da lógica já abordada no modo de operação está também presente no modo de configuração. Primeiro, verificamos se estamos neste estado pela primeira vez, se for o caso, troca o valor para zero e configura as flags de acordo.

Por fim, liga o LED L3 a amarelo e lê e guarda os ticks atuais.

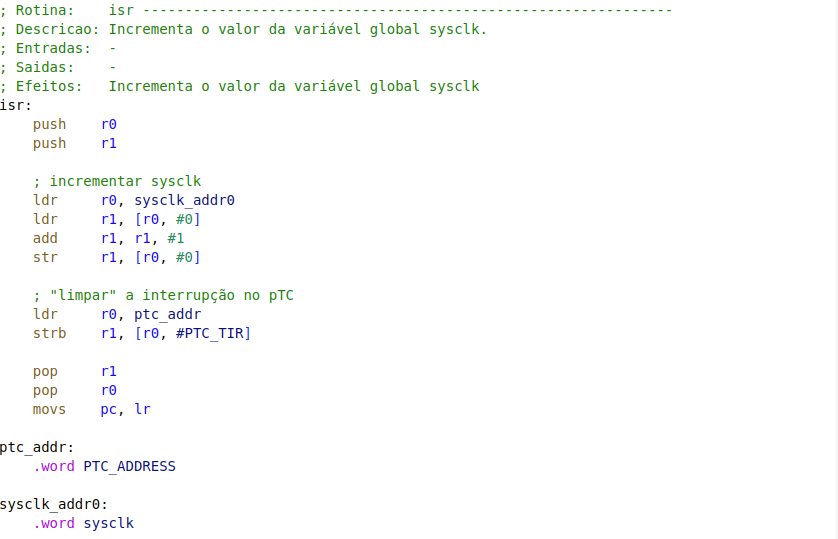
Se não for a primeira vez neste estado, lêmos o valor da *flag conf\_leds\_phase* para saber se temos de apagar ou acender os leds e operar sobre os mesmos. Se os leds tiverem na fase off, passamos o valor 0 em r0 e r1 para desligar os leds, se for para acender, em r0 passamos 3 (amarelo) e 2 em r1 (verde), chamando em seguida a rotina *config\_mode\_leds\_phases*.

No final chamamos a rotina *set\_crossing\_time* para guardar o valor da configuração do tempo de atravessamento do peão.



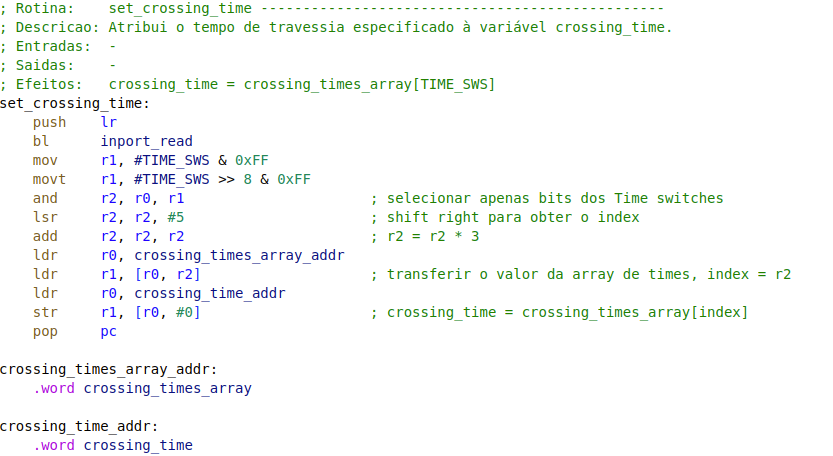
Nesta rotina, vamos ler o valor da *flag conf\_first\_time\_on\_phase* para ver se estamos neste estado pela primeira vez e iniciamos o *timer*, chamamos as funções respetivas para cada semáforo (pedestre ou de trânsito) com os devidos valores previamente fornecidos em r0 e r1. No fim, dizemos que não estamos neste estado pela primeira vez.

Novamente, se não for a primeira vez no estado, apenas verificamos se o tempo já terminou, se isso se verificar, invertemos o valor da *flag conf\_leds\_phase* para alternar de modo a que na próxima vez, se as luzes tiverem ligadas serem desligadas e vice-versa.

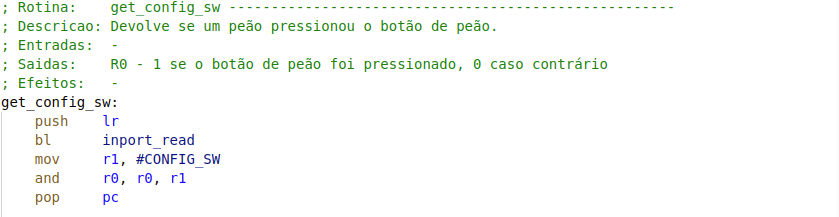


Rotina de atendimento às interrupções que incrementa o valor da variável sysclk usada para contar o tempo de 100 em 100 ms.

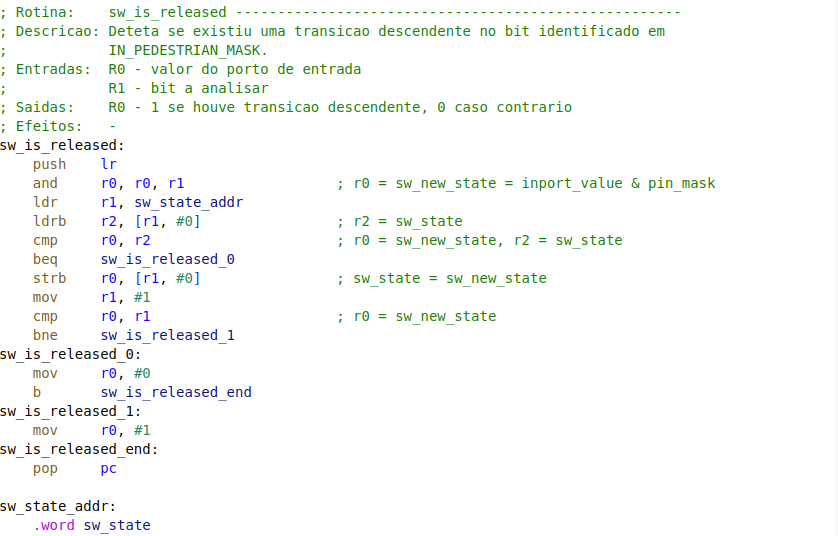
**Serviços**



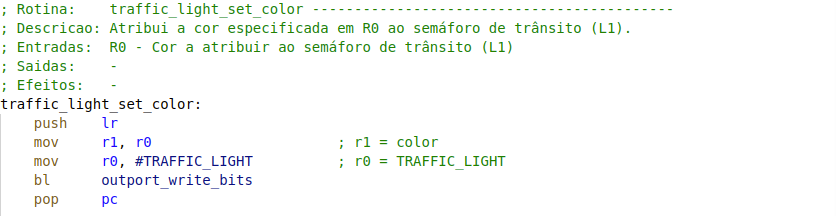
Rotina usada para estabelecer a temporização do *crossing time*. Lemos o valor do *inport* com a máscara dos 3 últimos bits (TIME\_SWS), *shiftamos* para a direita por 5 e com este valor como *index* do *array crossing\_times\_array*, vamos buscar a temporização pretendida e guardar o seu valor em crossing\_time, o qual é lido para calcular o tempo de atravessamento na rotina *operation\_mode*.



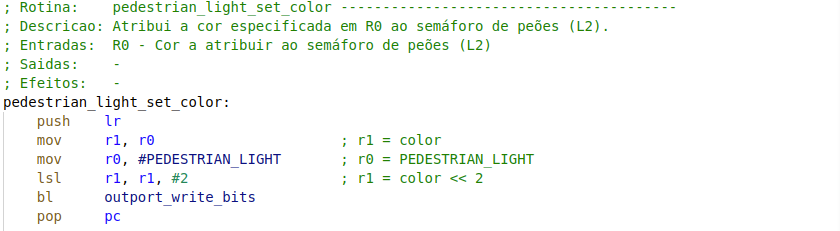
A rotina *get\_config\_sw* lê o valor do inport com a máscara do 5º bit afim de perceber o valor presente no *switch* de configuração.



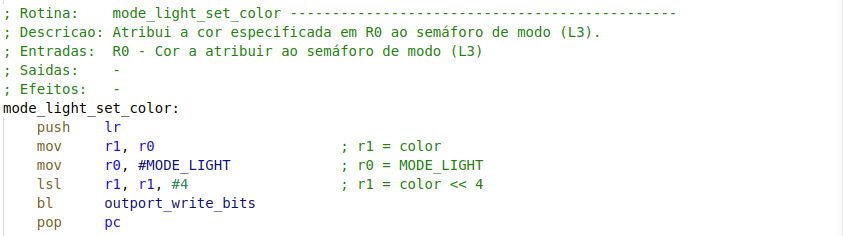
Rotina cuja função é detetar se existiu ou não uma transição descendente no bit que representa o botão do peão, retornando 1 em caso afirmativo e 0 em caso negativo.



Rotina que tem como objetivo escrever no outport a cor passada em r0 referente ao LED do trânsito



Rotina que tem como objetivo escrever no outport a cor passada em r0 e depois *shiftada* para a esquerda de 2 valores, referente ao LED dos peões.

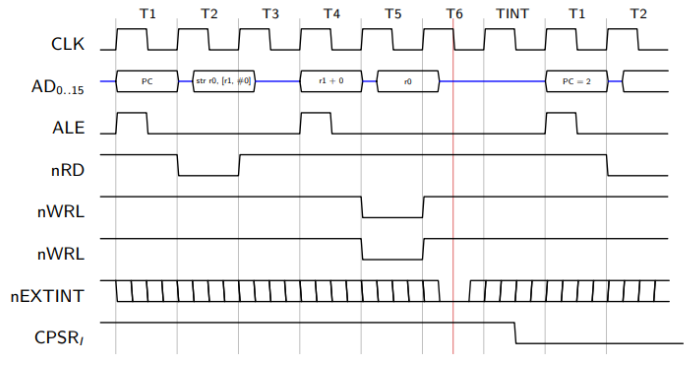


Rotina que tem como objetivo escrever no outport a cor passada em r0 e depois *shiftada* para a esquerda de 4 valores, referente ao LED que sinaliza o modo de operação (config ou operation).

Foram usadas as rotinas de inicialização e configuração dos periféricos dadas nas aulas, sem termos efetuado alterações às mesmas. Nomeadamente:

* *sysclk\_init*
* *sysclk\_get\_ticks*
* *sysclk\_elapsed*
* *inport\_read*
* *outport\_set\_bits*
* *outport\_init*
* *outport\_write\_bits*
* *outport\_write*
* *ptc\_init*
* *ptc\_start*
* *ptc\_stop*
* *ptc\_get\_value*
* *ptc\_clr\_irq*

**Latência máxima do sistema no atendimento dos pedidos de interrupção geradas pelo pTC**



A latência máxima no atendimento de interrupções ocorre quando o pedido é iniciado no inicio de uma intrução de acesso à memória, ou seja, 6 clocks da intrução, mais 1 clock de tratamento das ‘flags’ e colocação do PC em 0x0002, + 6 clocks de redirecionamento para a rotina ‘isr’ uma vez que a mesma não se encontra no endereço 0x0002.   
Uma vez que o clock do P16 funciona a uma frequencia de 50KHz (20 micro segundos), temos 79 x 20 = 260 microsegundos = 0,26 milisegundos.

**Tempo gasto, no pior dos casos, na execução da rotina ISR**

No pior dos casos, ou seja, instruções de acesso à memória (6 clocks), o tempo gasto na execução da rotina de atendimento a interrupções são 1,46 milisegundos.

|  |  |
| --- | --- |
| **Intrução** | **Número de clocks usados** |
| Instrução de acesso à memória | 6 |
| Tratamento das flags e PC = 0x0002 | 1 |
| ldr PC, isr\_addr | 6 |
| push R0 | 6 |
| push R1 | 6 |
| ldr R0, sysclk\_addr | 6 |
| ldr R1, [R0, #0] | 6 |
| add R1, R1, #1 | 3 |
| str R1, [R0, #0] | 6 |
| ldr R0, ptc\_addr | 6 |
| strb R1, [Ro, #PTC\_TIR] | 6 |
| pop R1 | 6 |
| pop R0 | 6 |
| movs PC, LR | 3 |
| TOTAL | 73 |

Uma vez que o clock do P16 funciona a uma frequencia de 50KHz (20 micro segundos), temos 73 x 20 = 1460 microsegundos = 1,46 milisegundos.

**Conclusões**

Na resolução deste trabalho prático, sentimos diversas dificuldades. Primariamente, na esquematisação do flowchart, o qual reconhecemos que elaboramos de uma forma complexa, cheia de flags para mudança e de estados. Teria ficado de mais fácil compreensão e por sua vez implementação em assembly se tivessemos usado “cases”.  
Outra dificuldade que sentimos foi ao tentar fazer “debug”, usando o p16sim, não é possível configurar o clock a 1KHz, ou seja, tinhamos de alterar o SYSCLK\_FREQ. Ao usar a placa SPD16, é impossível, fazer debug passo a passo com o atendimento a interrupções ligado, tendo para isso de desligar a ligação nINT\_EXT da placa e liga-la numa das saídas da ATB. No nosso caso, tinhamos um bug na rotina ISR, estavamos a usar ldrb e strb em vez de ldr e str, o qual por debug não conseguimos detetar, apenas a rever o código.

Consideramos que estas dificuldades fazem parte do processo de aprendizagem, as quais nos fizeram ser mais atentos e minuciosos no processo de experimentação. Apesar de tudo, fomos percistentes e conseguimos atingir os objetivos.

Este trabalho, contribuio para termos um conhecimento mais sólido do conteúdo da matéria da cadeira, sobretudo no que se refere aos tópicos usados neste trabalho, entrada e saída de dados, temporização, interrupções externas, organização do programa em rotinas e implementação de máquinas de estados em software.