**UNIVERSIDADE FEDERAL DE MINAS GERAIS**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

# Software Básico
# Trabalho Prático II – Interrupções e Escalonamento de Tarefas

**Caio Felipe Zanatelli**
**João Victor Tamm**

**Professor: Marcos Augusto M. Vieira**

Belo Horizonte
29 de novembro de 2017

**Sumário**

**1 Introdução** **2**

**2 Semáforo** **2**
2.1 Descrição . . . . . 2
2.2 Questões de Projeto . . . . . 2

**3 Escalonador de Tarefas** **3**
3.1 Descrição . . . . . 3
3.2 Questões de Projeto . . . . . 3

**4 Conclusão** **5**

**5 Referências Bibliográficas** **5**

## 1 Introdução

O objetivo principal do trabalho prático desenvolvido é conhecer o funcionamento de um microcontrolador e o escalonamento de processos, assim lidando com diversas funcionalidades, entre elas o tratamento de interrupções advindas de um *switch button* e um temporizador. De modo a facilitar a familiarização com o dispositivo utilizado (o *kit* de desenvolvimento *LaunchPad MSP430*) e com as técnicas que deveriam ser utilizadas para o desenvolvimento deste trabalho, alguns documentos e *templates* foram disponibilizados pelo professor da disciplina. Com o intuito de introduzir os conceitos aos poucos, o trabalho foi dividido em duas partes, sendo a primeira delas referente ao tratamento de interrupções e a segunda parte referente ao desenvolvimento de um escalonador de tarefas. As etapas deste trabalho, bem como suas respectivas decisões de implementação, serão descritas nas seções seguintes.

## 2 Semáforo

Esta seção tem como objetivo apresentar o desenvolvimento da primeira parte do trabalho prático em questão, isto é, um simulador de semáforo implementado em ambiente embarcado através do microcontrolador MSP430, abordando tópicos como a descrição geral do problema e questões de projeto tomadas.

### 2.1 Descrição

Nesta etapa da implementação foi proposta a simulação de um semáforo fazendo uso dos LED's embutidos no *kit* de desenvolvimento. Assim, o LED0 corresponde à luz vermelha do sinal, o LED1 à verde e, devido ao fato de existirem apenas dois LED's acoplados ao ambiente disponível, a luz amarela é representada pelas luzes verde e vermelha acesas simultaneamente.

De modo a introduzir o conceito e o funcionamento de interrupções em ambiente real, foi utilizado um temporizador e um *switch button*. O temporizador é utilizado para a contagem do tempo que compõe a configuração do semáforo abordada anteriormente, sendo responsável por definir os tempos de cada luz do semáforo de acordo com cada estado. O *switch*, por sua vez, foi utilizado para simular a ação de um pedestre que deseja atravessar a rua, fazendo com que a interrupção gerada ao pressionar o botão reconfigure os tempos das luzes caso condições que serão abordadas na sequência sejam atendidas. O funcionamento de tal semáforo se dá da seguinte maneira:

1. Comportamento sem interferência do pedestre:

    - A luz vermelha deve ficar acesa durante 2 segundos;
    - A luz amarela deve ficar acesa durante 1 segundo;
    - A luz verde deve ficar acesa durante 8 segundos.

2. Comportamento com interferência do pedestre:

    - Se a luz verde estiver acesa, seu tempo será decrementado em 5 segundos, entretanto, caso o tempo restante para a luz verde seja inferior a 3 segundos, este é mantido.

### 2.2 Questões de Projeto

O método utilizado para o desenvolvimento das técnicas apresentadas acima foi bem simples. Primeiramente foram setados os pinos, portas de entrada e saída, o pino de interrupção e o timer. Em seguida foram criadas duas funções para tratar as interrupções envolvidas.

A primeira delas corresponde ao timer, sua função é basicamente controlar o tempo em que cada um dos LED's deve ficar aceso. Para que seja possível, uma variável auxiliar foi criada. Tal variável é incrementada a cada iteração. Quando atinge um valor igual a 2 segundos a luz verde acende, em seguida ao chegar a 10 segundos a vermelha é acesa indicando que o semáforo está no estado amarelo, por fim ao atingir 11 segundos a luz verde apaga e o contador é resetado. O outro procedimento trata as interrupções causadas pelo clique do botão, ou seja, simula o pedestre que deseja atravessar a rua. A função simplesmente verifica se o tempo necessário para que o sinal abra para o pedestre é superior a 3 segundos, caso seja o tempo é incrementado em 5 unidades de forma a reduzir o tempo de espera para o pedestre. Para desenvolver tal algoritmo foi usado um template obtido através da página do professor da disciplina.

## 3 Escalonador de Tarefas

Esta seção tem como objetivo apresentar a segunda etapa do trabalho prático em questão, referente ao desenvolvimento de um escalonador de tarefas utilizando o *kit* de desenvolvimento *LaunchPad MSP430*.

### 3.1 Descrição

Como já citado, a última etapa deste trabalho visa o desenvolvimento de um escalonador de tarefas em um ambiente microcontrolado, com o intuito de intensificar os conceitos de troca de contexto e interrupções vistos em sala de aula. Tal sistema deve executar três tarefas simultaneamente, são elas:

- **Tarefa 1**: Piscar a luz vermelha do *launchpad* a cada 2 segundos.
- **Tarefa 2**: Piscar a luz verde do *launchpad* a cada 10 segundos.
- **Tarefa 3**: Inverter os tempos das tarefas 1 e 2 caso o botão seja pressionado.

O escalonador funciona da seguinte forma: inicialmente são criadas três tarefas e três filas para armazenar os estados de cada tarefa. Assim, cada fila é utilizada para armazenar o PC, SR e os registradores $r4$ até $r15$. O escalonamento ocorre através de uma interrupção gerada pelo temporizador. Com isso, quando essa interrupção ocorrer os registradores da tarefa atual devem ser salvos. Além disso, o registrador $r1$, que contém a pilha de execução, deve ser atualizado para a próxima tarefa. Para essa nova tarefa que será executada, os registradores devem ser recuperados. Vale ressaltar, neste ponto, que o escalonamento utilizado implementa a técnica *Round-Robin*, isto é, cada tarefa é executada por um período de tempo pré-estabelecido, tempo este igual para todas as tarefas, ocorrendo a troca de tarefas quando o tempo da tarefa atual é finalizado, efetuando a devida troca de contexto citada anteriormente.

### 3.2 Questões de Projeto

As técnicas utilizadas nesta etapa do trabalho tiveram a necessidade de ser um pouco mais elaboradas, uma vez que a complexidade foi bem superior quando comparada à primeira etapa. Para obter sucesso no escalonamento das tarefas foi necessário o domínio de alguns conceitos, como o salvamento e a restauração de contextos e a executação tarefas, além do entendimento de interrupções.

Primeiramente, doze registradores foram criados – de forma a possibilitar a realização das trocas de contexto –, três pilhas, sendo cada uma delas responsável por uma tarefa, e um vetor de ponteiros encarregado de armazenar o *Stack Pointer* de cada uma das pilhas de execução.

Além disso, de modo a facilitar o controle em relação à troca de tarefas, foram criadas duas *flags* auxiliares, uma para salvar o identificador da tarefa que está sendo executada no momento atual e outra para identificar se o botão foi pressionado, implementado através de uma interrupção, uma vez que no momento em que o botão for pressionado pode ser que a tarefa sendo executada não seja a tarefa 3, o que não surtiria o efeito desejado. De modo a solucionar todos os problemas propostos na especificação do trabalho prático em questão, houve a necessidade de criar diversas funções, algumas delas em *assembly*. Tais funções serão discutidas na sequência.

O primeiro ponto a ser discutido está relacionado às funções que foram implementadas em *assembly*. Os procedimentos foram *load* e *store* do *stack pointer*, que simplesmente lidam com o *stack pointer* no *array* de *stack pointers*; e *saveContext* e *restoreContext*, os quais são responsáveis pelo salvamento e restauração do contexto com o auxílio dos registradores citados anteriormente, proporcionando a troca de contextos sem perda de informação relativa a cada tarefa.

A função *task1* foi criada para tratar a *tarefa 1* proposta, a qual é responsável por acender um LED a cada 2 segundos, sendo o LED selecionado através de uma mascará de *bits* contida na variável *led2*, sendo seu valor configurado como o LED vermelho inicialmente. De modo a gerar o *delay* apropriado para que decorra 2 segundos para cada transição do estado do LED, a função *delay_ cycles(x)* foi utilizada, sendo o parâmetro $x$ um inteiro de 16 *bits* que identifica o número de ciclos de *clock* que serão aguardados. Para isso, o tempo foi calculado da seguinte forma: a frequência base utilizada para o cálculo foi 1MHz, isto é, 1000000 ciclos por segundo e, como são três tarefas, temos que cada uma delas utilizará 333333 ciclos. Entretanto, como este valor não pode ser armazenado em um inteiro de 16 *bits*, o valor utilizado foi 33333, o qual é ajustado por um fator de 10, isto é, para gerar um segundo são utilizadas 10 instruções *delay_ cycles(33333)*. Com isso, a fórmula construída para encontrar a quantidade de *delays* necessárias para a contagem de $n$ segundos, considerando que cada *delay* é executado sobre 33333 ciclos, é a seguinte:

$$Quantidade_{Delays} = n \times 10 \times delay(33333) \tag{1}$$

.

Além disso, por motivos de segurança, uma vez que *loops*, por serem *branches*, podem sofrer algum tipo de otimização por parte do compilador e acarretar *delays* maiores ou menores do que o esperado, as instruções de *delay* foram executadas uma a uma, isto é, um total de 100 instruções para gerar um *delay* de 10 segundos e um total de 20 instruções para gerar um *delay* de 2 segundos. Ainda, análoga à função *task1*, citada anteriormente, o procedimento *task2* refere-se à *tarefa 2*, a qual é responsável por acender um LED a cada 10 segundos, sendo o LED selecionado atravées de uma máscara de *bits*, indicada pela variável *led10*, sendo seu valor inicialmente configurado como o LED verde. As estratégicas para gerar o *delay* necessário foram as mesmas supracitadas, sendo a Equação 1 utilizada para isso.

O procedimento *task3*, por sua vez, é responsável pela implementação da *tarefa 3*, rotina esta que deve ser estudada com um pouco mais de cuidado. Para que a troca entre os tempos dos LED's ocorra (papel da função em questão), é necessário que haja uma interrupção no botão do *launchpad*. Assim, como existem três tarefas sendo escalonadas, a probabilidade de que a *task3* esteja sendo executada no momento em que o botão é pressionado é de 33,3%. Tendo isso em vista, verificar a *flag* de interrupção configurada para o *switch button* no interior da rotina referente à *task3* não é uma boa opção, uma vez que se a *task3* não estiver sendo executada no momento em que o botão é pressionado a troca de tarefas pode não ocorrer. Para resolver este problema, foi criada uma *flag* auxiliar e uma rotina de interrupção para tratar a ação de interrupção para o *switch*. Dessa forma, quando o botão é pressionado, a interrupção é executada e a *flag* auxiliar

é configurada com nível lógico alto, indicando que o botão foi pressionado. Quando a *task3* for executada, ela verifica se a *flag* auxiliar foi configurada e, caso positivo, a inversão das *tasks* é realizada e a *flag* auxiliar é então configurada com nível lógico baixo. Dessa forma, independente do momento em que o botão for pressionado, tal ação será visível para a *task3*, uma vez que, com esta abordagem, é sempre possível identificar se o botão foi pressionado ou não antes da execução da *task3*, e então efetuar a ação desejada.

## 4 Conclusão

Este trabalho apresentou um tópico de extrema relevância e ascensão para a Ciência da Computação: os sistemas embarcados. Através desses sistemas, é possível desenvolver uma variedade de aplicações que estão extremamente presentes na vida de todos os cidadãos, tais como eletrodomésticos, semáforos, elevadores, carros, entre vários outros. Com o crescimento da *Internet* das Coisas (IoT) e o consequente barateamento dos componentes de *hardware*, este assunto tem ficado cada vez mais em evidência, levando ao elevado investimento nesta área, como evidencia o surgimento de várias *start ups* neste ramo ao redor do mundo.

Neste contexto, este trabalho se mostrou bastante importante para a consolidação de conceitos teóricos vistos em sala de aula, uma vez que o tema escolhido está diretamente relacionado com toda a ementa do curso. Um ponto a se destacar está relacionado à fixação dos procedimentos de tratamento de interrupções e de escalonamento de tarefas, que foi possível através da realização deste trabalho, um conteúdo demasiadamente teórico e pouco explorado a nível prático em aulas sobre o tema. Além disso, o presente trabalho permitiu aos alunos envolvidos a familiarização e o aprendizado de novos conceitos em teoria de sistemas embarcados, o que foi possível através do estudo prático com o *kit* de desenvolvimento MSP430, fabricado pela *Texas Instruments*.

## 5 Referências Bibliográficas


Tanenbaum, Andrew S.; Austin, Todd.. Structured Computer Organization, 6th ed. Elsevier, 2005.

Texas Instrument. MSP430G2 LaunchPad Development Kit – User's Guide. Disponível em: <http://www.ti.com/lit/ug/slau318g/slau318g.pdf>. Acesso em 28 nov 2017.

Software Básico. Materiais de aula relativos à disciplina Software Básico no contexto da UFMG, 2º semestre de 2017. Disponível em: <http://homepages.dcc.ufmg.br/~mmvieira/se/se.html>. Acesso em: 29 nov 2017.