Trabalho 2: Sistema de *software* do Uóli

MC404 – Organização Básica de Computadores e Linguagem de Montagem

31 de Outubro de 2018

1 Introdução

Neste segundo trabalho da disciplina, você irá desenvolver todas as camadas de *software* responsáveis pelo controle do robô Uóli. Essas camadas, ilustradas na Figura 1, são divididas em três subcamadas: (a) Sistema Operacional UóLi (SOUL), (b) Biblioteca de Controle (BiCo) e (c) Lógica de Controle (LoCo).

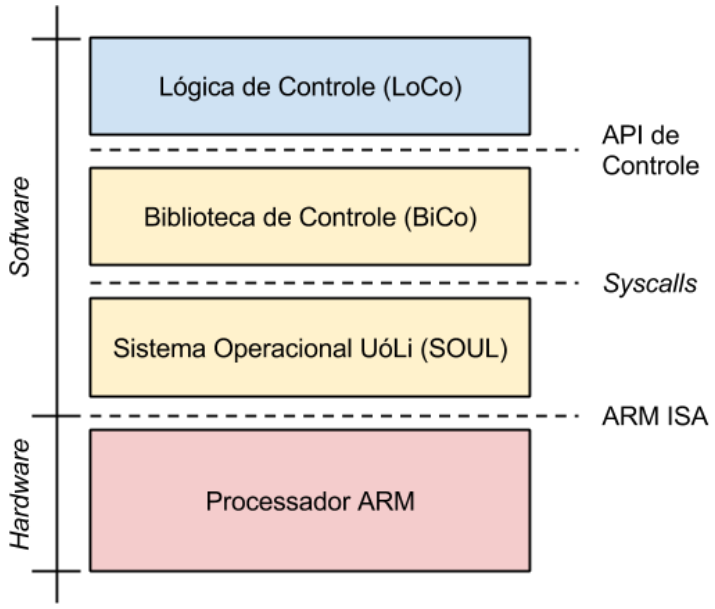

Figura 1: Pilha de *software* do robô Uóli.

A subcamada SOUL é responsável pelo gerenciamento do *hardware*, incluindo a configuração do *hardware* e o tratamento de interrupções de *hardware* e de *software*. Além disso, o SOUL deve prover um conjunto de serviços para a subcamada BiCo através de chamadas de sistemas, ou *syscalls*. A subcamada SOUL contém código que será executado em modo supervisor e deve ser implementada em linguagem de montagem.

A subcamada BiCo é responsável por prover uma interface de programação amigável para a Lógica de Controle, a API de Controle. A subcamada BiCo

também deve ser implementada em linguagem de montagem, mas seu código será executado no modo usuário e ligado com o código da subcamada LoCo com o auxílio do ligador (*linker*).

A subcamada LoCo é responsável pela lógica de controle do robô e deve invocar as funções definidas pela API de Controle e implementadas pela BiCo. A subcamada LoCo deve ser implementada em código na linguagem C e seu código será executado no modo usuário. Como informado acima, o código da LoCo deverá ser ligado ao código da BiCo com o ligador.

2 Subcamada LoCo

Como descrito acima, o código da subcamada LoCo deve ser implementado em linguagem C e deve fazer uso das rotinas disponíveis na API de Controle para enviar comandos para o robô. A API está descrita no arquivo "`api_robot.h`"

Você desenvolverá dois programas na linguagem C para a camada LoCo: o `segue-parede.c` e o `ronda.c`.

2.1 Lógica de controle do programa `segue-parede.c`

A lógica de controle do programa `segue-parede.c` deve ter dois modos de operação: (a) busca-parede e (b) segue-parede.

(a) A lógica busca-parede é iniciada assim que o Uóli é ligado. Esta lógica deve fazer o Uóli andar em linha reta até se aproximar de uma parede (o Uóli não deve colidir com a parede). Após encontrar a parede, o Uóli deve ajustar sua posição de forma que a parede fique do lado direito dele.

(b) Uma vez que a posição foi ajustada, o modo segue-parede deve ser ativado. Neste modo, o Uóli deve andar para frente acompanhando a parede, ou seja, sempre mantendo a parede à sua direita, ajustando o traçado à medida que a parede se distanciar ou ficar muito próxima do robô. Novamente, é importante que o Uóli não colida com as paredes do ambiente.

2.2 Lógica de controle do programa `ronda.c`

Para o programa `ronda.c`, a lógica de controle de seu robô deve fazer o robô realizar uma ronda no ambiente. Para realizar a ronda, seu programa deve:

(a) Fazer o robô andar em uma "espiral quadrada". Para isso, o robô deve andar um pouco para a frente, fazer uma curva de **aproximadamente** 90 graus para a direita, andar mais um pouco para a frente, fazer outra curva para a direita, e assim por diante. É importante que, após cada curva, a distância percorrida para frente seja um pouco maior. A distância deve ser ajustada em unidades de tempo do sistema (veja abaixo). Seu robô deve ser configurado para andar para frente por uma unidade de tempo até a primeira curva à direita, depois andar por 2 unidades de tempo para frente antes de realizar a próxima curva, e assim por diante, até atingir 50 unidades de tempo. A partir daí o robô deve iniciar uma nova ronda.

(b) Sua lógica deve verificar os sensores para garantir que o robô não colida com as paredes. Caso haja uma parede no traçado do robô, você deve ajustar o curso do robô girando-o para a direita, certificando-se de que ele não colida com a parede.

3 Subcamada BiCo

O código da subcamada BiCo deve implementar as rotinas da API de Controle em linguagem de montagem do ARM. A API está descrita no arquivo "`api_robot.h`". Para controlar o *hardware*, o código deve realizar chamadas ao sistema, ou *syscalls*. As *syscalls* são definidas na Seção 4.3.

4 Subcamada SOUL

A subcamada SOUL deve gerenciar o *hardware* do sistema e prover serviços para a subcamada BiCo através das chamadas de sistemas.

4.1 Tempo do sistema (*System time*)

O SOUL possui um relógio interno que mantém o tempo do sistema, ou *System Time*. O tempo do sistema deve ser iniciado com 0 sempre que o sistema for (re)iniciado e o tempo deve ser incrementado de 1 unidade a cada `TIME_SZ` ciclos do relógio (*clock*) de periféricos (similar ao que será desenvolvido na atividade de laboratório 9). O símbolo `TIME_SZ` deve ser definido como uma constante utilizando-se a diretiva `.set` no arquivo que gerencia as interrupções do GPT. Utilize um valor razoável para que o tempo do sistema não passe muito rápido nem muito devagar.

4.2 Atendendo a chamadas de sistema (*syscalls*)

Na atividade de Laboratório 9 você implementará um pequeno programa em linguagem de montagem do ARM para atender às interrupções de *hardware* do tipo IRQ. Nesse trabalho, você deve modificar e expandir a implementação anterior para atender também às **interrupções de *software*** (*syscalls*), disparadas pela instrução SVC.

Na convenção do ARMv7 (ABI - *Application Binary Interface*), para realizar uma chamada ao sistema, coloca-se o número da *syscall* no registrador `R7`, e os parâmetros seguem a mesma convenção de uma chamada de função comum (devem estar nos registradores `R0` a `R3`); o valor de retorno é passado via registrador `R0`. Esta será a convenção adotada neste trabalho.

Para realizar a chamada de sistema, o código de usuário utiliza a instrução `svc 0x0`. Esta instrução irá gerar uma interrupção por *software* (também conhecida como *trap*) e fará o registrador `PC` apontar para a posição `base_vet + 0x08`, em que `base_vet` é a base do vetor de interrupções que é definido na seção `.iv`. Nesse ponto, então, o processador troca o modo para SUPERVISOR - a implementação de uma chamada de sistema, portanto, é similar à implementação de interrupções de *hardware*. No entanto, você recebe em `R7` um valor que corresponde ao número da *syscall* que se deseja chamar. O seu tratador

de chamadas de sistema deve, portanto, analisar o valor contido nesse registrador e selecionar a rotina de tratamento adequada (`set_motor_speed`, ou outra). Lembre-se de que, para retornar do tratador de chamadas de sistema para o código do usuário que invocou a *syscall*, você deve utilizar a seguinte instrução especial:

```
movs pc, lr
```

Essa instrução, além de retornar ao código que estava sendo executado antes da interrupção, recupera o registrador `CPSR` original, modificando o modo do processador para o modo corrente antes da ocorrência da interrupção. (**O sufixo "s"é necessário**)

4.3 Descrição das *Syscalls*

A Tabela 1 apresenta as *syscalls* que devem ser implementadas. Caso ocorra mais de um erro na execução da chamada de sistema, o código retornado deve ser o do erro de **maior valor**.

Syscall	Parâmetros	Retorno
`read_sonar` (**Código: 21**)	`R0`: Identificador do sonar (Valores válidos: 0 a 15)	`R0`: Valor obtido na leitura dos sonares; -1, caso o identificador do sonar seja inválido
`set_motor_speed` (**Código: 20**)	`R0`: Identificador do motor (valores válidos: 0 ou 1); `R1`: Velocidade do motor (valores válidos: 0 a 63)	`R0`: -1 caso o o motor seja inválido; -2, caso a velocidade seja inválida; 0, caso OK
`get_time` (**Código: 17**)	-	`R0`: retorna o tempo do sistema.
`set_time` (**Código: 18**)	`R0`: tempo do sistema	-

Tabela 1: Chamadas de sistemas (*syscalls*) do Sistema Operacional Uóli.

4.4 Iniciando o Sistema

Ao iniciar o sistema, o SOUL deve realizar duas atividades: (1) configurar o *hardware* e (2) transferir a execução para a aplicação de controle no modo usuário.

4.4.1 Configurando o *Hardware*

Na atividade de laboratório Laboratório 9 você terá trechos de código para configurar o GPT e o TZIC. Nesse trabalho, você deverá expandir seu código para configurar o GPIO. Essas alterações vão permitir que o SOUL execute os serviços oferecidos através das *Syscalls*.

O dispositivo de propósito geral de entradas e saídas do sistema, ou GPIO, tem como função prover uma interface para conexão de componentes externos, como periféricos de um robô, ao processador.

Ao todo, o GPIO do simulador fornece 32 pinos configuráveis que foram conectados a dispositivos periféricos do robô. As conexões foram realizadas de acordo com a Tabela **??**.

Pino do GPIO	**Conexão no periférico**	**Configuração do GPIO**
0	FLAG	Entrada
1	TRIGGER	Saída
2	SONAR_MUX[0]	Saída
3	SONAR_MUX[1]	Saída
4	SONAR_MUX[2]	Saída
5	SONAR_MUX[3]	Saída
6	SONAR_DATA[0]	Entrada
7	SONAR_DATA[1]	Entrada
8	SONAR_DATA[2]	Entrada
9	SONAR_DATA[3]	Entrada
10	SONAR_DATA[4]	Entrada
11	SONAR_DATA[5]	Entrada
12	SONAR_DATA[6]	Entrada
13	SONAR_DATA[7]	Entrada
14	SONAR_DATA[8]	Entrada
15	SONAR_DATA[9]	Entrada
16	SONAR_DATA[10]	Entrada
17	SONAR_DATA[11]	Entrada
18	MOTOR0_WRITE	Saída
19	MOTOR0_SPEED[0]	Saída
20	MOTOR0_SPEED[1]	Saída
21	MOTOR0_SPEED[2]	Saída
22	MOTOR0_SPEED[3]	Saída
23	MOTOR0_SPEED[4]	Saída
24	MOTOR0_SPEED[5]	Saída
25	MOTOR1_WRITE	Saída
26	MOTOR1_SPEED[0]	Saída
27	MOTOR1_SPEED[1]	Saída
28	MOTOR1_SPEED[2]	Saída
29	MOTOR1_SPEED[3]	Saída
30	MOTOR1_SPEED[4]	Saída
31	MOTOR1_SPEED[5]	Saída

Tabela 2: Mapeamento dos pinos configuráveis do GPIO nos periféricos.

As portas do GPIO estão mapeadas no espaço de endereçamento físico do sistema. O dispositivo está conectado ao barramento do sistema no endereço base **0x53F84000**. Ele possui três registradores de 32 *bits*: DR, GDIR e PSR. O endereço associado a cada registrador está indicado na Tabela 3.

A seguir são descritos cada um dos registradores:

1. **DR: Registrador de dados (*Data Register*)**

Registrador	Deslocamento
DR	0x53F84000
GDIR	0x53F84004
PSR	0x53F84008

Tabela 3: Registradores do GPIO.

Este registrador armazena os dados que serão direcionados aos pinos de saída ou os dados que vieram dos pinos de entrada para o GPIO, dependendo de como o pino esteja programado no registrador GDIR (descrito a seguir). O resultado de uma leitura deste registrador depende de como os *bits* do GDIR estão configurados.

- Se o n-ésimo *bit* do GDIR (GDIR[n]) possuir o valor lógico igual a '1', então o retorno da leitura do *bit* DR[n] será o último valor escrito pelo *software* em DR[n].
- Se o *bit* GDIR[n] possuir valor lógico igual a '0', então o retorno da leitura do *bit* DR[n] será o valor lógico do sinal de entrada no n-ésimo pino de entrada do GPIO.

2. **GDIR: Registrador de direções (*Direction Register*)**

   Este registrador controla as direções de cada um dos pinos de conexão do GPIO.

   - Se o GDIR[n] possuir valor lógico igual a '0', o n-ésimo pino está configurada como entrada.
   - Se o GDIR[n] possuir valor lógico igual a '1', o n-ésima pino está configurada como saída.

3. **PSR: Registrador de estado Pad (*Pad Status Register*)**

   Este é um registrador disponível apenas para leitura. Cada *bit* armazena o valor do sinal do pino de entrada correspondente.

   - PSR[n] retorna o valor lógico do sinal do n-ésimo pino.

4.5 Transferindo a execução para a aplicação de controle

Como visto acima, após a configuração do *hardware*, o SOUL deve transferir o fluxo de execução para a aplicação de controle. Lembre-se de que a aplicação de controle deve ser executada no modo usuário, dessa forma, o SOUL deve modificar o modo de operação para USER.

Além de ajustar o modo de operação, o SOUL deve configurar uma pilha para o processo da aplicação de controle.

5 Apêndice - *Hardware*

Esta seção apresenta o funcionamento básico do *hardware* criado para o robô, tanto a parte eletrônica, quanto os aspectos necessários para o controle e programação destes periféricos.

5.1 Sonares

Como visto em laboratórios anteriores, o robô dispõe de 16 sonares, sendo 8 frontais e 8 traseiros. Cada um destes sonares trata-se de um dispositivo HC SR04, que é controlado via dois sinais, o ***TRIGGER*** e o ***ECHO***. Basicamente, o *TRIGGER* é um sinal lógico que quando levado de um nível baixo para um nível alto, mantendo este por pelo menos 10 ms, e novamente colocando em um nível baixo, faz o sonar obter uma nova leitura.

A leitura retornada pelo sonar é dada pela quantidade de tempo durante a qual o sinal de *ECHO* fica no nível alto. Dessa forma, podemos obter um valor proporcional para a distância, baseado em uma fórmula disponibilizada pelo fabricante do dispositivo.

Para facilitar a comunicação com estes 16 sonares, um novo *hardware* foi proposto. O esquemático deste *hardware* é ilustrado na Figura 2.

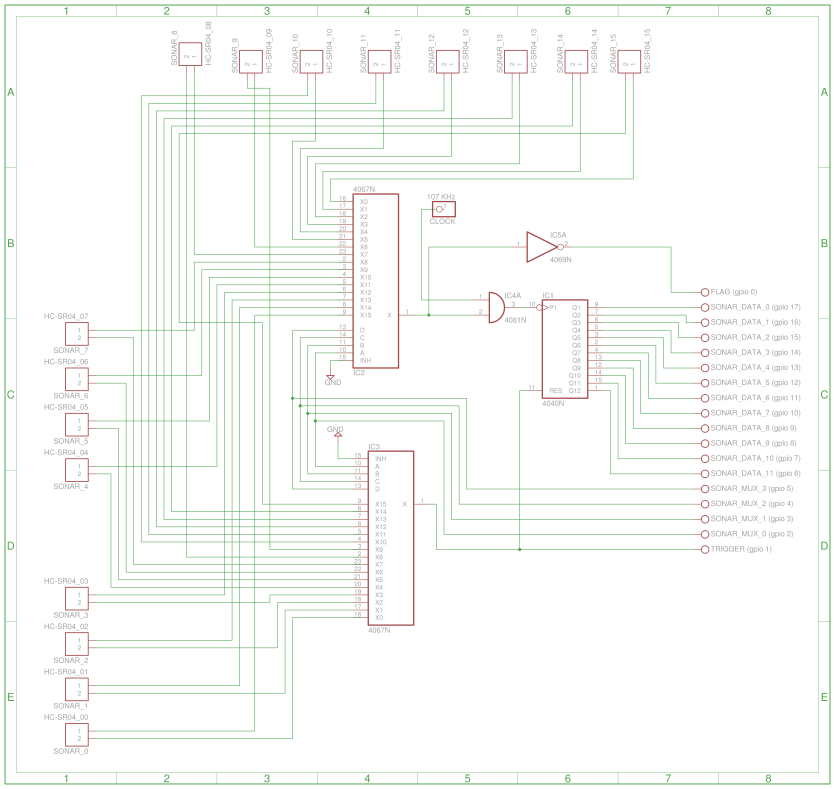

Figura 2: Esquemático do *hardware* de comunicação com os sonares

Como o acesso aos sonares é feito individualmente, o circuito de controle conta com um multiplexador e um de-multiplexador, para ambos os casos utilizamos o circuito integrado (CI) 74HC4067. Estes dois CIs são identificados no esquemático como IC3 para o multiplexador e IC2 para o de-multiplexador.

Outro CI importante para o funcionamento do circuito é o contador binário MC14040B, que no momento em que o TRIGGER é levado para o nível lógico '1', zera sua contagem. Quando o sinal de *ECHO* for para o nível lógico '1', ele inicia sua contagem, quando este sinal retorna para '0', a contagem é parada.

Dessa forma temos na saída deste contador, pinos `SONAR_DATA[11:0]`, um valor de 12 *bits* proporcional à distância encontrada pelo sonar. A frequência de *clock* para este contador é calibrada para nunca zerar sua saída durante o momento que o *ECHO* for '1', mesmo que o sonar selecionado não encontre obstáculo.

Para auxiliar na obtenção da informação, um sinal adicional é colocado, informando quando o contador terminou sua contagem, trata-se do sinal de *FLAG*. Sempre que uma leitura for solicitada, após o sinal de *TRIGGER* retornar para '0', basta esperar o momento que o sinal de *FLAG* vá para '1', garantindo que a leitura foi completada.

O fluxograma abaixo nos dá uma visão mais clara de como podemos realizar uma leitura via algum dos sonares (Figura 3).

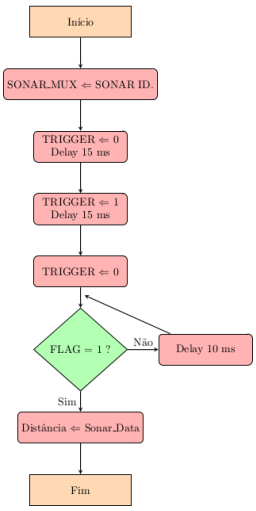

Figura 3: Fluxograma para leitura dos sonares

5.2 Motores

O robô possui dois motores, um para cada roda. O controle de velocidade para estes motores é implementado usando PWM. Um *hardware* específico é proposto para realizar o controle do PWM, no qual para cada motor fornecemos 6 *bits* para quantificar a velocidade e mais um *bit* responsável por habilitar a escrita dessa nova velocidade. O esquemático pode ser encontrado na Figura 4.

Para gerar o PWM, utilizamos o CI LTC6992, que converte um valor analógico, de 0 V até 1 V, para uma largura de pulso proporcional. Para obtermos o valor analógico correspondente, utilizamos o conversor digital-analógico TLC7226, no qual os 6 *bits* vão mapear o PWM e o *bit* de *write*, quando colocado em '0', habilita o novo valor.

6 Entrega e Avaliação

- **23-11-2018, até as 23h:59min** – Fator Multiplicativo: 1.0

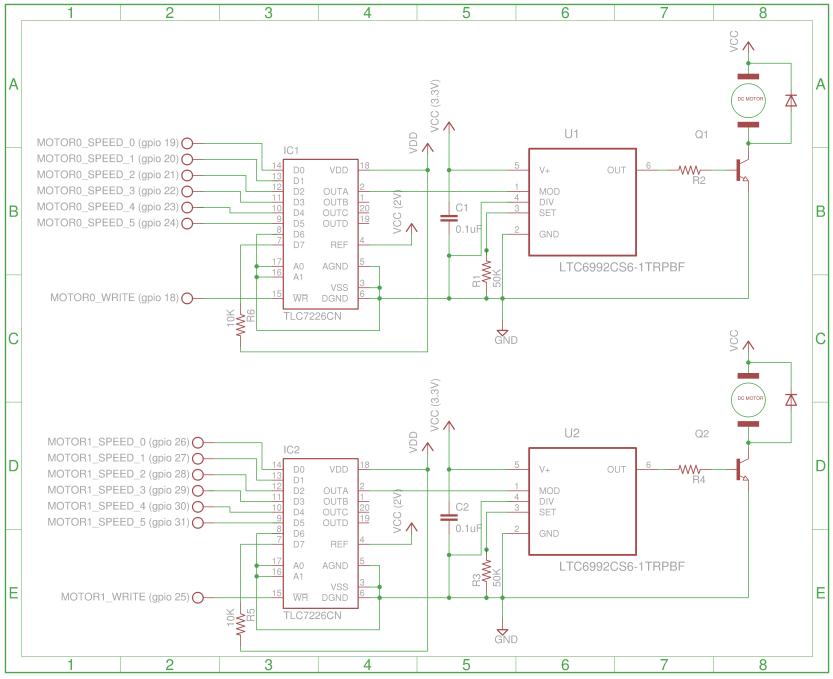

Figura 4: Esquemático do *hardware* de motores

- **24-11-2018, até as 23h:59min** – Fator Multiplicativo: 0.8
- **25-11-2018, até as 23h:59min** – Fator Multiplicativo: 0.6
- Utilize a plataforma SuSy para entrega: Turmas A e B, Turmas E e F
- Os trabalhos podem ser individuais ou em dupla. Para os trabalhos feitos em dupla: **ambos os alunos devem submeter os mesmos arquivos!**
- Deve ser entregue **apenas** um arquivo de nome `raXXXXXX.tar.gz` (máximo: 2MB), que por sua vez deve conter um diretório raXXXXXX que inclua todos os arquivos de código do seu trabalho, um arquivo chamado `grupo.txt`, e os dois Makefiles: `Makefile-ronda` e `Makefile-segue-parede`
- Ambos os Makefiles devem gerar o arquivo `disk.img` como regra padrão.
- Utilize a flag `-f` para selecionar o Makefile apropriado. Ex: `make -f Makefile-ronda`
- O arquivo `grupo.txt` deve conter o RA dos integrantes da dupla, ou apenas um RA, no caso de trabalho individual. Os valores devem ser da forma raZZZZZZ e devem ser separados por uma quebra de linha; nenhum outro dado deve ser colocado nesse arquivo.
- Seu código deve estar bem documentado, incluindo a descrição das rotinas e trechos de código não triviais.
- O arquivo worlds_mc404.zip contém os cenários de teste.

- Utilizem o grupo de discussão para dúvidas e esclarecimentos! (https://groups.google.com/forum/#!forum/unicamp-mc404-2s2018)
- **Qualquer tentativa de fraude implicará média 0 na disciplina, para todos os envolvidos.**

7 Perguntas Frequentes, Dicas e Outras Dúvidas

- O código de usuário começará no endereço `0x77812000`. O sistema operacional deve desviar o fluxo de execução para este endereço, para invocar o código de usuário.
- Não há necessidade de tratar erros de *syscalls* inexistentes (diferentes da que foram definidas neste trabalho).
- Interrupções por *software* no ARM estão sempre habilitadas. Não é necessário fazer nada para habilitá-las.
- Como inicializar a pilha para diferentes modos?
  Mude para o modo desejado (usando a instrução `msr`/`mrs`) e inicie o registrador `SP`. Lembre-se que cada modo de operação do registrador, possui bancos de registradores diferentes. Para mudar de modo, os 5 *bits* menos significativos do CPRS devem conter os valores mostrados na Tabela 4.

M[4:0]	Mode	Register Set
10000	User	`R0-R14, CPSR, PC`
10001	FIQ	`R0-R7, R8_fiq-R14_fiq, CPSR, SPSR_fiq, PC`
10010	IRQ	`R0-R12, R13_irq, R14_irq, CPSR, SPSR_irq, PC`
10011	SVC (supervisor)	`R0-R12, R13_svc R14_svc CPSR, SPSR_irq, PC`
10111	Abort	`R0-R12, R13_abt R14_abt CPSR, SPSR_abt PC`
11011	Undefined	`R0-R12, R13_und R14_und, CPSR, SPSR_und PC`
11111	System (ARMv4+)	`R0-R14, CPSR, PC`

Tabela 4: Modos de Operação

  Mais informações podem ser obtidas clicando aqui (Slides do ARM ISA e Instruções `mrs, msr` – Slide 66), ou clicando aqui (Sobre Registrador de Status).
- Lembre-se que você deve escolher um valor apropriado para a pilha de cada modo de operação. Para organização, você pode usar a diretiva `.set` para definir estes endereços.
- Para carregar o vetor de interrupções, utilize a instrução
  `mcr p15, 0, r0, c12, c0, 0 @ r0, endereço base do vetor`.
  Mais informações sobre esta instrução: acesse o link, clicando aqui.