Universidade Federal de Ouro Preto

Instituto de Ciência Exatas e Aplicadas

Departamento de Engenharia Elétrica

Laboratório de Microprocessadores e Microcontroladores – CEA580

Prática 2

Nome: Graziele de Cássia Rodrigues

Matrícula: 21.1.8120

Objetivos

Compreender o processo de transferência de dados usando os modos de endereçamento ARM;

Utilizar diretivas ARM para criar áreas, definir constantes e declarar variáveis.

Referências

Mazidi and Naimi “The STM32F103 Arm Microcontroller and Embedded Systems”, Cap. 2.

Installing the Keil for STM32F10x step by step tutorial

STM32 Assembly Programming in Keil step by step tutorial

Lista de Materiais

Placa STM32F103C8 Blue Pill

ST-Link V2

Keil IDE, disponível em http://www.keil.com/

Keil.STM32F1xx_DFP.2.4.0.pack, disponível em https://www.keil.com/dd2/pack/

Atividade 1

Criar um projeto usando o software Keil para executar o algoritmo Ex1.s enviado em anexo.

Explique as diferenças entre as duas áreas definidas no algoritmo.

No código do Ex1 temos as seguintes áreas:

```
    AREA OUR_PROG, CODE, READONLY
__main
```

Nesta área, denominada "OUR_PROG", está sendo definida uma seção de código (CODE) que é somente leitura (READONLY), ou seja, utiliza a memória flash e portanto o código desta seção não pode ser modificado durante a execução do programa.

Nesta área temos o rótulo __main que indica o ponto de entrada do programa, o ponto a partir do qual a execução começará.

```
AREA    OUR_DATA, DATA, READWRITE
```

Na área denominada "OUR_DATA", está sendo definida uma seção (DATA) que é legível e gravável (READWRITE), ou seja, está sendo usada a memória RAM.

Nesta área estão definidas três variáveis chamadas "A," "B," e "C" usando a diretiva SPACE e alocando espaço na memória para essas variáveis com o valor de 4 bytes de espaço. Essas variáveis são usadas para armazenar valores e realizar cálculos.

Partindo, então para análise do código:

Em LDR R0, =A armazenamos o valor 0x20000000 que indica o endereço inicial de A.

Após isso, é realizada a instrução STR R1, [R0] que armazena o valor atual em R1 na memória no endereço especificado (inicial) por R0.

Em MOV R1, #0xA, STR R1, [R0] está movendo o valor A para o registrador R1 e também armazenando o valor no endereço especificado por R0.

Após isso temos o seguinte:

```
LDR     R0, =B
        MOV     R1, #4
        STR     R1, [R0]
        MOV     R1, #0x8
        STR     R1, [R0]
```

Nessa parte é “gravado” o endereço inicial de B em R0 que terá o valor agora de 0x2000004, salvo em R1 o valor 4 e carregado para o valor no endereço especificado por R0. O mesmo é feito mas com 0x8.

Os valores do endereço foram acrescentados de 4 em 4 por conta da definição das labels como 4.

Logo em seguida, é feito

```
LDR     R0, =A
        LDR     R1, [R0]

        LDR     R0, =B
        LDR     R2, [R0]
```

Em R0 é colocado o endereço de A e colocado em R1 o conteúdo armazenado em R0. E posteriormente o endereço de B é colocado em R2 de R0.

Por fim, temos as instruções que realizam a soma de 0x8 e A e armazenado no endereço inicial de R0. O valor final será 0x00000012, ou seja, 18.

```
ADD     R3, R1, R2
        LDR     R0, =C
        STR     R3, [R0]
```

Atividade 2

No código para atividade dois, temos as seguintes áreas definidas

```
        AREA OUR_PROG, CODE, READONLY
__main

  AREA  OUR_DATA, DATA, READONLY
```

Ambas as áreas são do tipo READONLY, ou seja, é permitido apenas leituras e usado memória flash. A primeira área é do tipo CODE, ou seja código, a segunda área é do tipo DATA, ou seja, armazenamento.

Na área OUR DATA é usado DCD para alocar e inicializar dados na memória, sendo uma word (32 bits).

E após LDR R0, =A é carregado em R0 o conteúdo do endereço A que é 0x5

Observação, não seria possível executar a instrução STR R1, [R0] sem que nada fosse alterado nesse algoritmo, pois nossa AREA de data é apenas do tipo leitura, além disso, foi utilizado uma atribuição ao inicializar um dado na memória com o DCD, não seria possível sobrescrever e seria necessário usar a diretiva SPACE.

Após a execução de cada instrução teremos:

R1, [R0]	0X5
R2, [R0]	0X2
R3, [R0]	0X3

ADDS R1, R1, R2	0X7
ADDS R1,R2,R3	0XA

O algoritmo para armazenar resposta final em SRAM seria

```
        EXPORT __main

        AREA OUR_PROG, CODE, READONLY
__main
        LDR     R0, =A
        LDR     R1, [R0]

        LDR     R0, =B
        LDR     R2, [R0]

        LDR     R0, =C
        LDR     R3, [R0]

        ADDS    R1,R1,R2
        ADDS    R1,R1,R3

        LDR R0, =D
        STR R1, [R0]

loop    B   loop

  AREA  OUR_DATA, DATA, READWRITE
A  DCD  0x5
B  DCD  0x2
C  DCD  0x3
D SPACE 4
  END
```

Atividade 3

As diretivas EQU e RN.

A diretiva EQU é usada para definir um valor constante associado a um nome. No algoritmo, toda vez que tiver alguma referência de A,B ou C será definido o valor especificado, ou seja, para "A," ele a substituirá por 0x12345678; para "B," ele substituirá por 0x2; e para "C," ele substituirá por 0x3.

Por sua vez, a diretiva RN é usada para renomear um registrador ou um endereço de memória. No nosso algoritmo, quando houver um IN1, IN2, IN3 e OUT o compilador tratará respectivamente como R1, R2, R3 e R4.

Utilizar essas diretivas são vantajosas para simplificar o código, deixando-o mais legível e entendível, visto que não é necessário modificar o código todo quando for realizar uma alteração.

Como a diretiva EQU apenas associa o nome A a um valor constante, mas não aloca espaço na memória para armazenar esse valor, ainda sim foi necessário utilizar o LDR para carregar o conteúdo em IN1. Foi utilizado LDR, pois está sendo alocado 32bits (8 bytes), caso fosse utilizado LDRB seria alocado apenas 1byte e apresentaria um comportamento incorreto.

No código, está sendo utilizado a memória flash, ou seja, realizando apenas leituras e é movido 0 para OUT com o objetivo de garantir que se inicia sem nada e os cálculos serem usado corretamente.

Após cada instrução tem-se o seguinte nos registradores:

LDR IN1,=A	Após a execução desta instrução, o registrador IN1 conterá o valor 0x12345678 (o valor associado a A)
MOV IN2,#B	Após a execução desta instrução, o registrador IN2 conterá o valor 0x2 (o valor associado a B)
MOV IN3,#C	Após a execução desta instrução, o registrador IN3 conterá o valor 0x3 (o valor associado a C)
ADDS OUT,OUT,IN1	soma de 0 e 0x12345678 (valor em IN1), que resulta em 0x12345678
ADDS OUT,OUT,IN2	soma de 0x12345678 e 0x2 (valor em IN2), resultando em 0x1234567A
ADDS OUT,OUT,IN3	soma de 0x1234567A e 0x3 (valor em IN3), resultando em 0x1234567D

Para que seja possível armazenar a resposta final do cálculo na memória SRAM do micro o código ficará:

```
AREA OUR_PROG, CODE, READONLY
__main
   LDR  IN1,=A
   MOV  IN2,#B
   MOV  IN3,#C
   MOV  OUT,#0
```

```
   ADDS OUT,OUT,IN1
   ADDS OUT,OUT,IN2
   ADDS OUT,OUT,IN3

  ;armazenamento sram
   LDR  R5, =SRAM_ADDRESS ; Carrega o endereço da SRAM em R5 (defina SRAM_ADDRESS)
   STR  OUT, [R5]       ; Armazena o valor de OUT na SRAM

loop
   B   loop

AREA SRAM_DATA, DATA, READWRITE
SRAM_RESULT SPACE 4
```