ELTD13z
Laboratório
Microcontroladores/Microprocessadores

Prática_01a2

Prof. Enio R. Ribeiro

Universidade Federal de Itajubá - UNIFEI

1. Sistema de desenvolvimento com microcontrolador

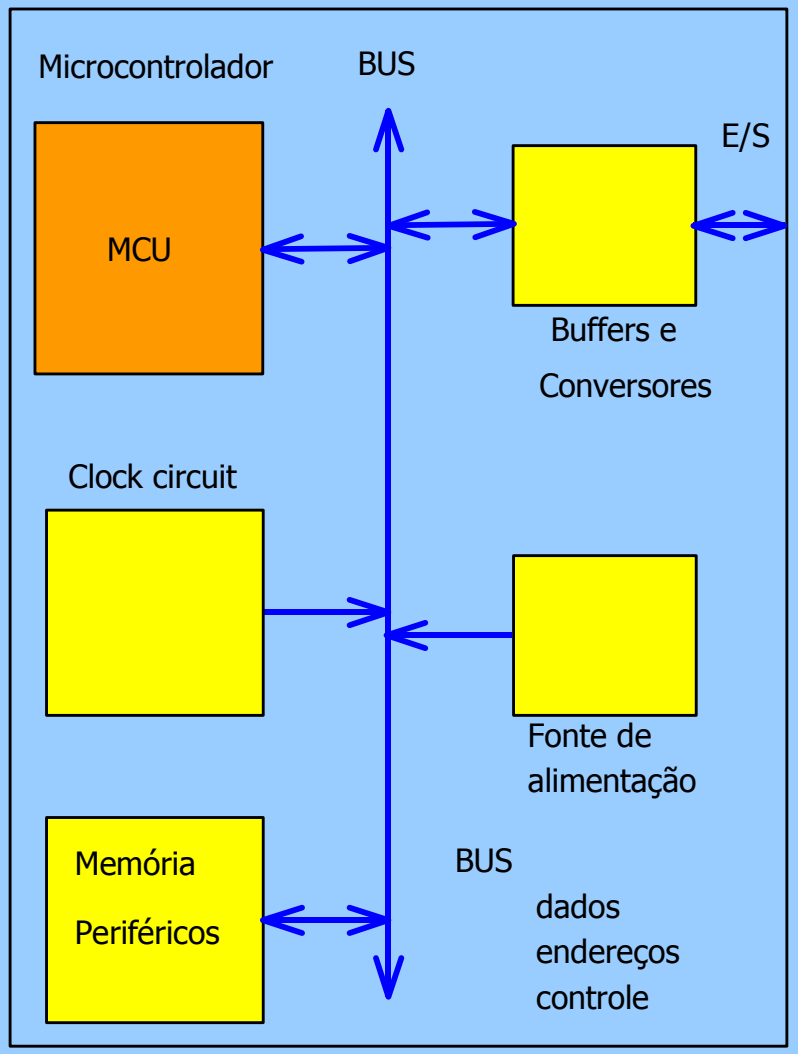


Fig. 1.a – Diagrama em blocos: sistema microcontrolado.

1. Sistema de desenvolvimento com microcontrolador

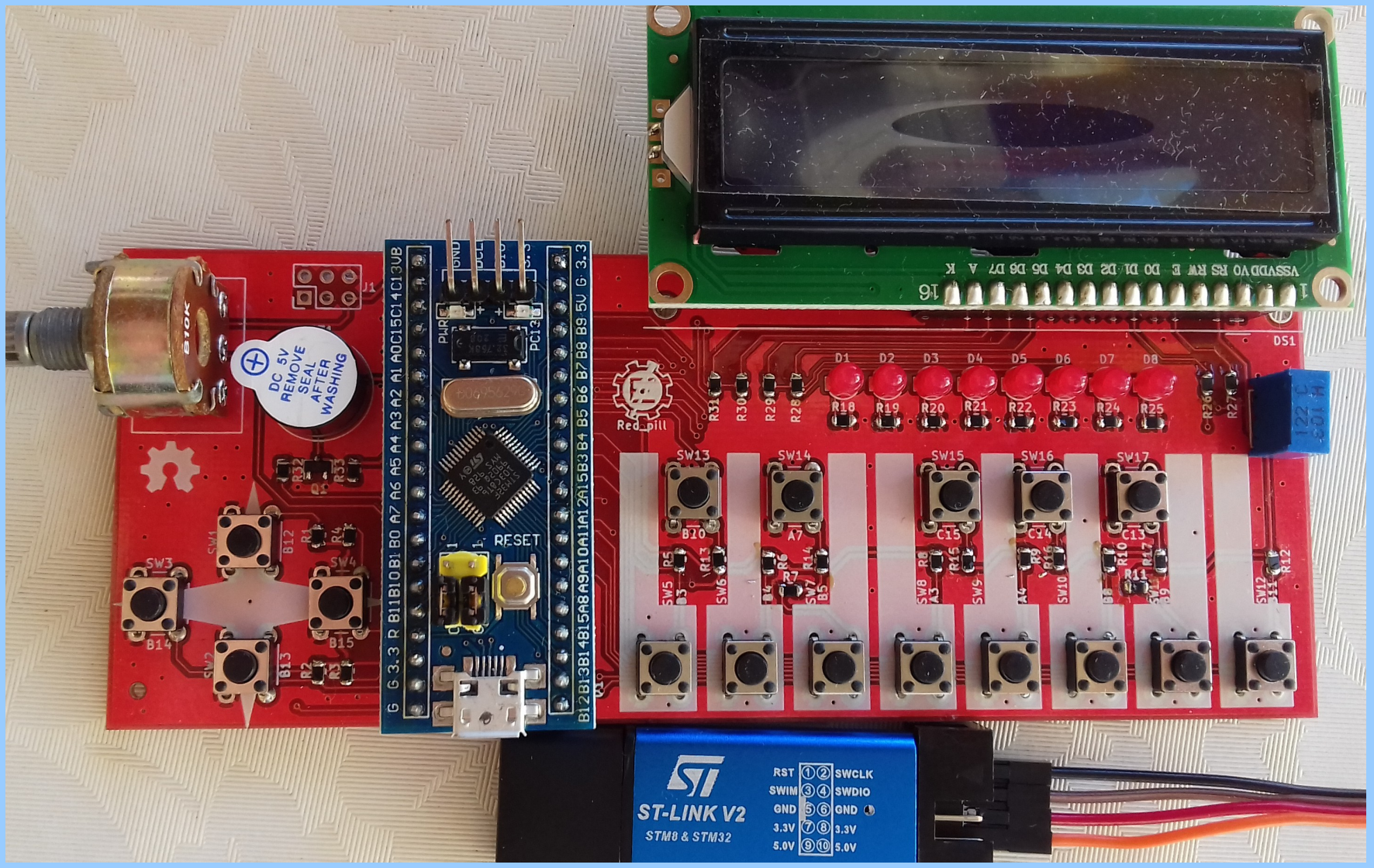


Fig. 1.b – Sistema de desenvolvimento microcontrolado baseado no STM32F103C8T6

2. Microcontrolador: STM32F103x

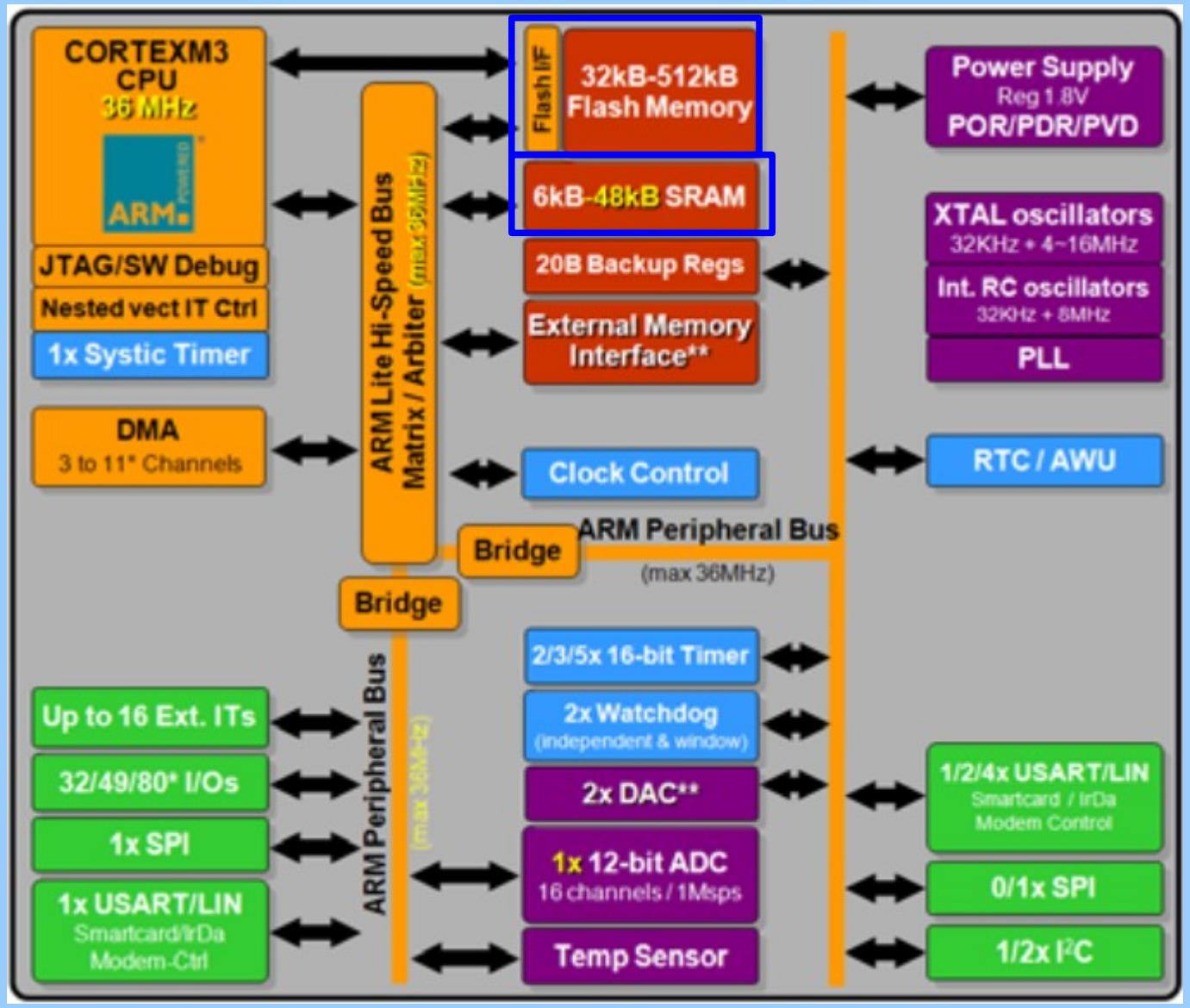


Fig. 2 – Diagrama em blocos (simplificado) do microcontrolador STM32F103x – origem: InsideCORTEX-1221142709.pdf.

2. Microcontrolador: STM32F103x

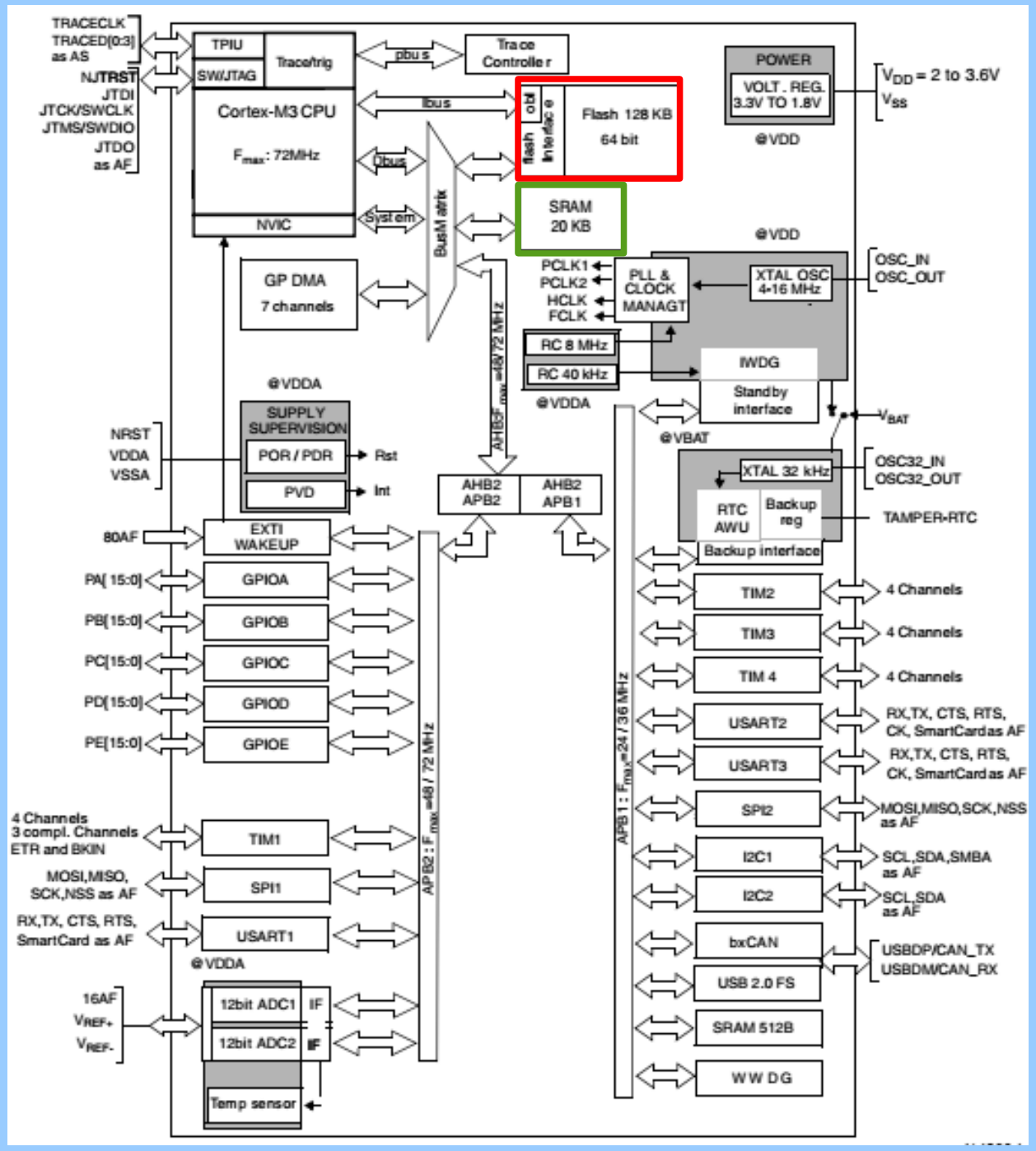


Fig. 3 – Diagrama em blocos (expandido) do microcontrolador STM32F103x.

2.1 Mapa de memória: STM32F103x

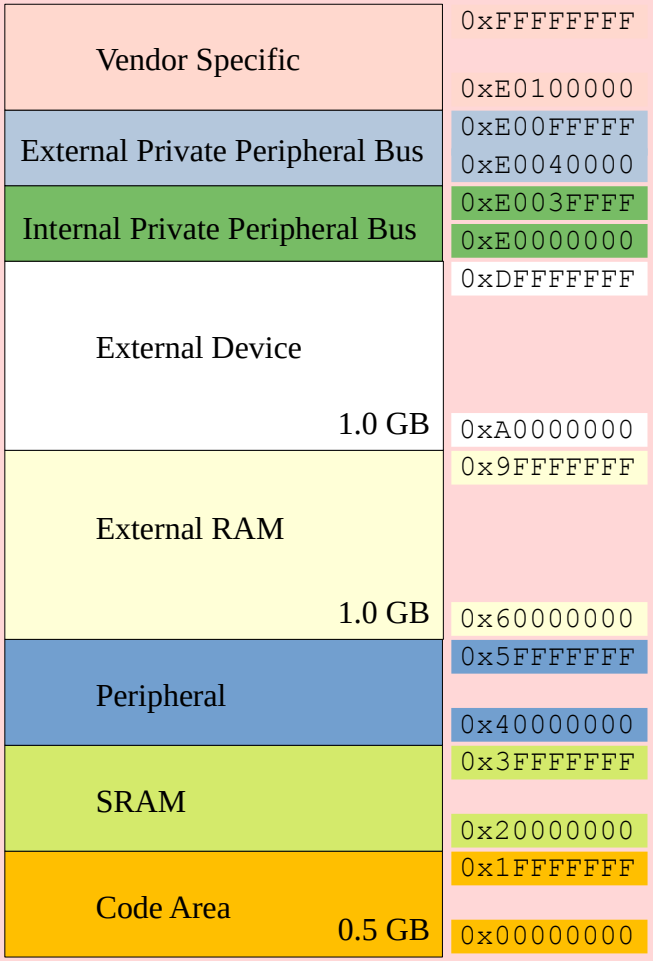


Fig. STM32Fx Memory map – baseada no manual: ddi0337g_cortex_m3_r2p0_trm

2.1 Mapa de memória: STM32F103x

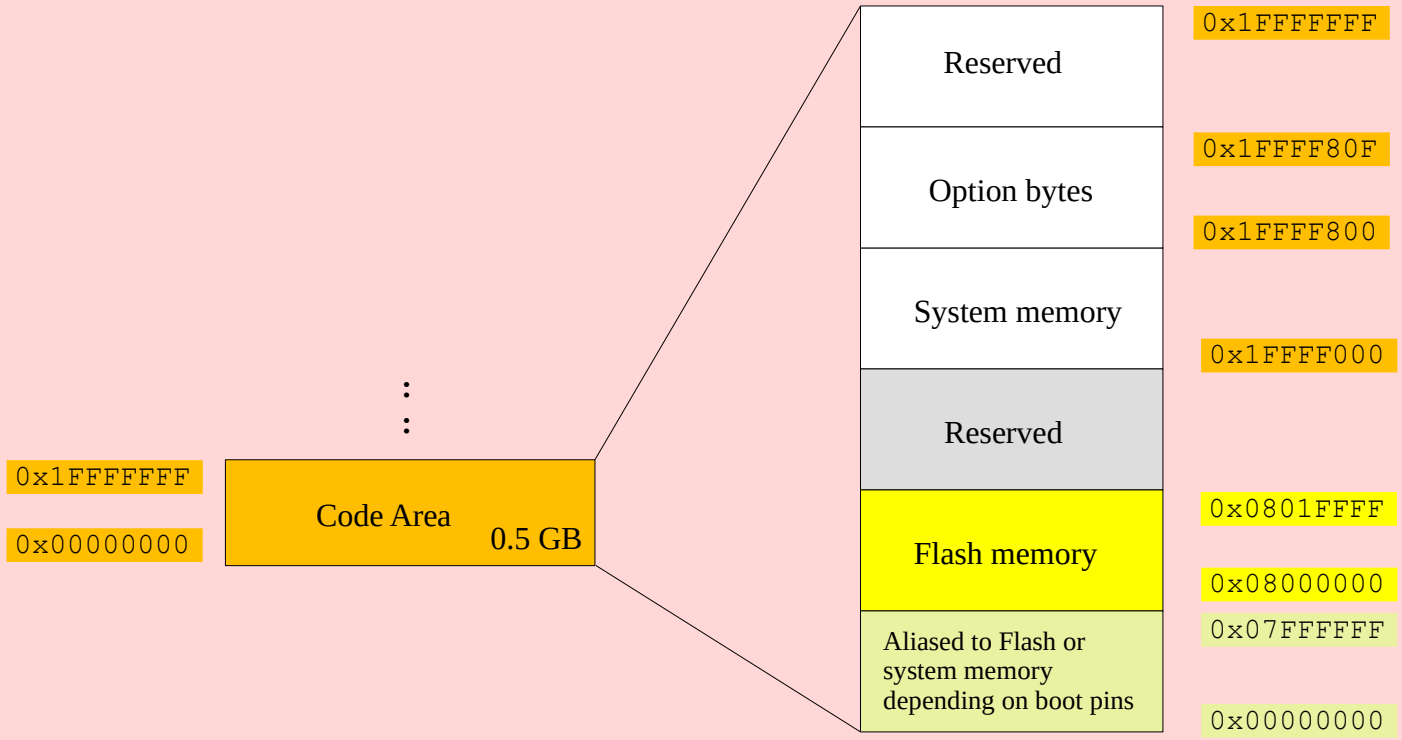


Fig. STM32Fx Memory map – baseada no manual: stm32f103c8.pdf

3.1 Memória e construção de programa

a) Organização

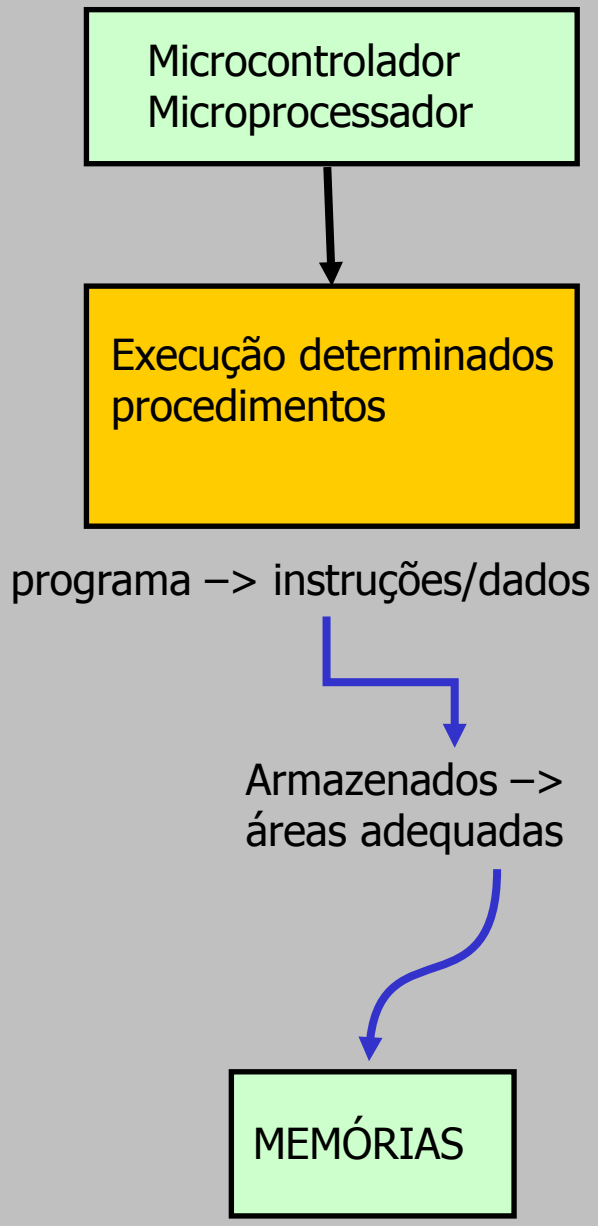
Dados –> armazenados –> números binários
(representação numérica: binário, decimal, hexadecimal)

Bit –> menor unidade dado binário –> 0, 1

Byte –> 8 bits –> quantidade de bits posição de memória

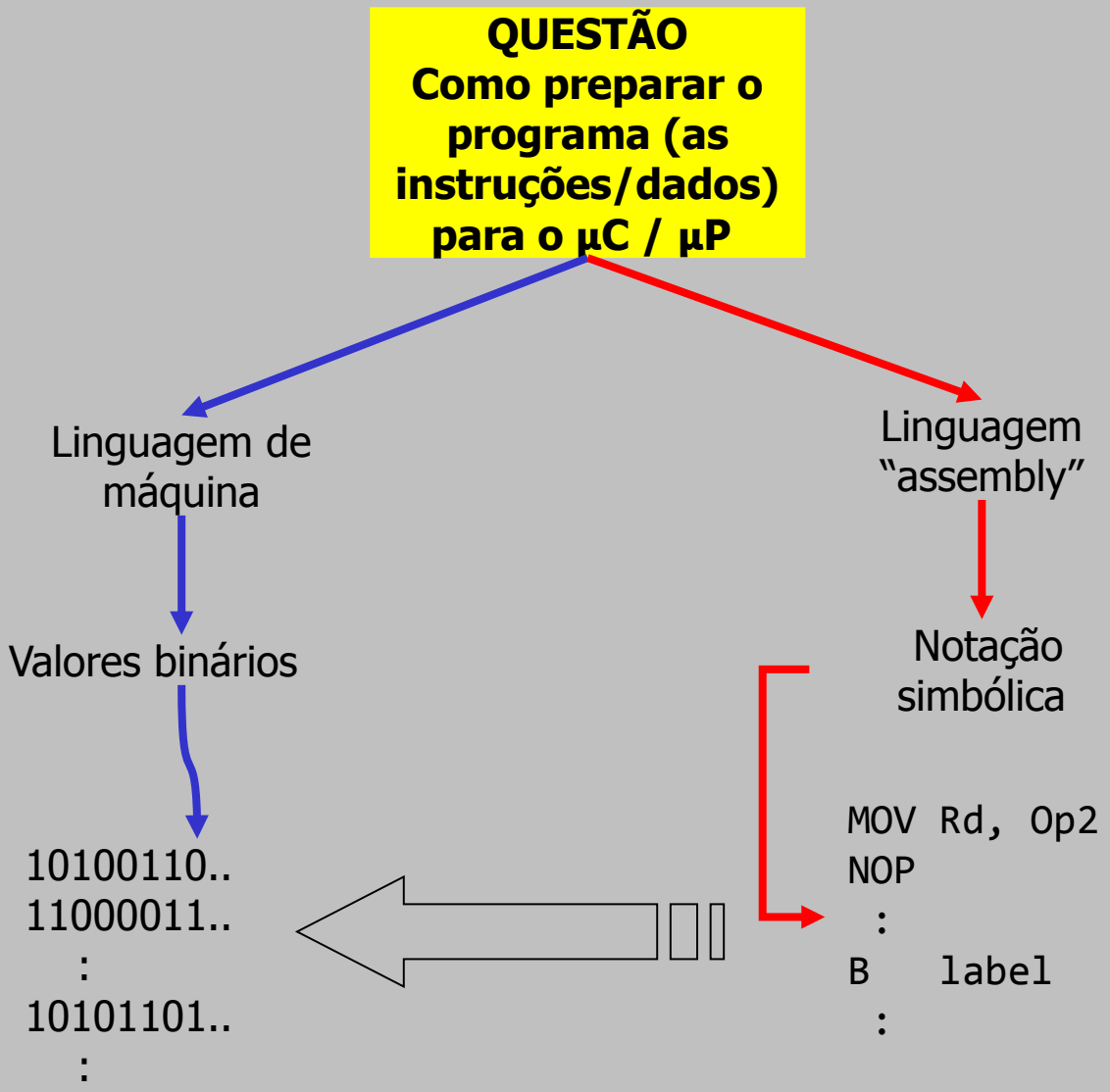
Half Word –> 16 bits (2 bytes)

Word –> 32 bits (endereçamento STM32F1x)



3.2 Programa: como fazer

3.2a Programa: como fazer



3.2b Programa: como fazer

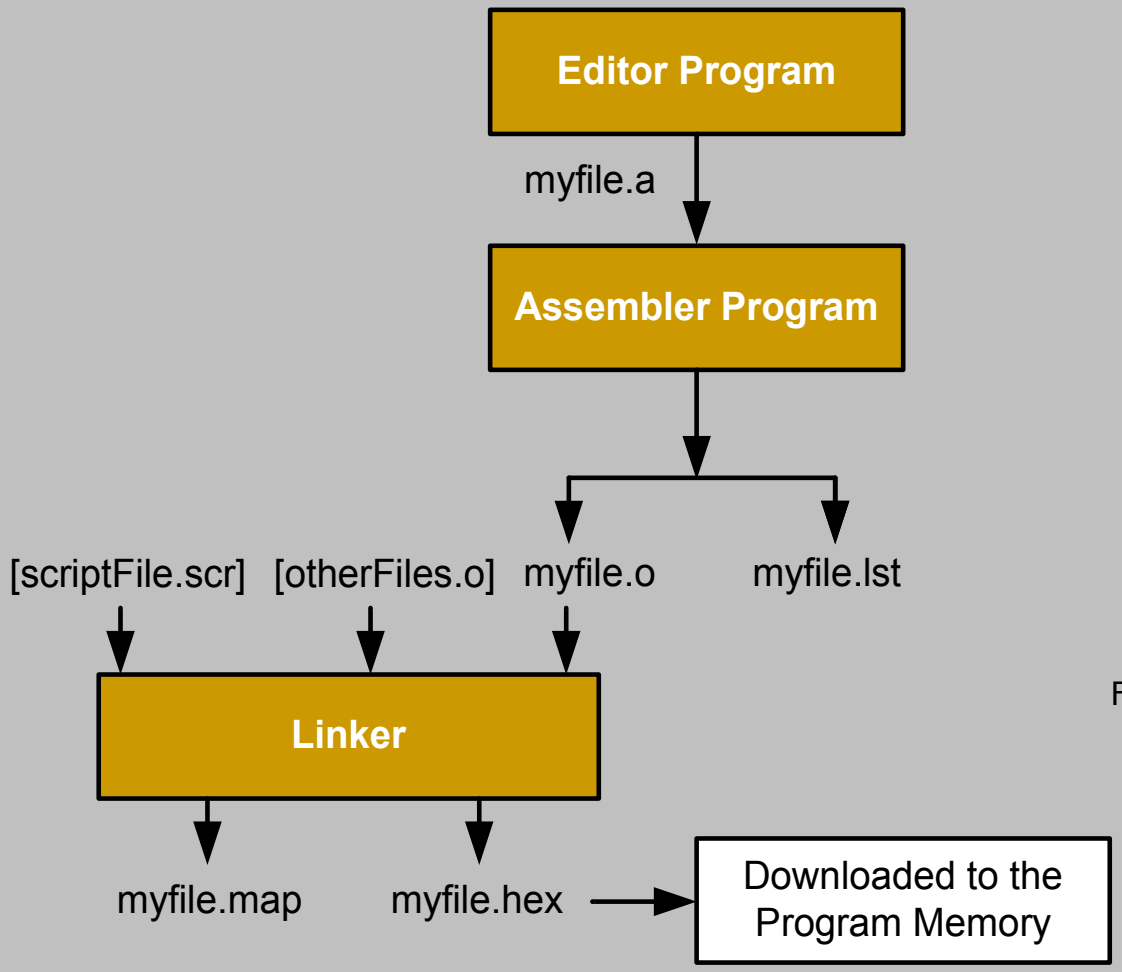


Fig. Fluxograma 1 – https://nicerland.com/stm32f103/

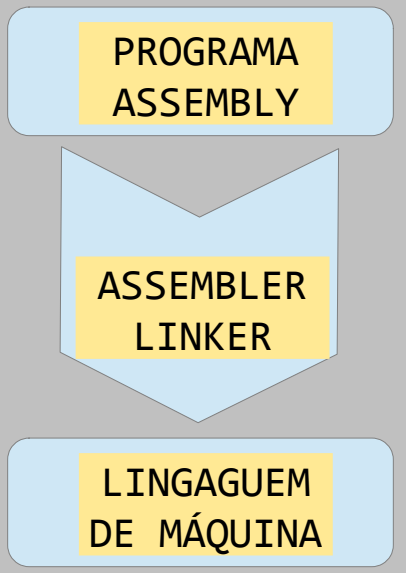
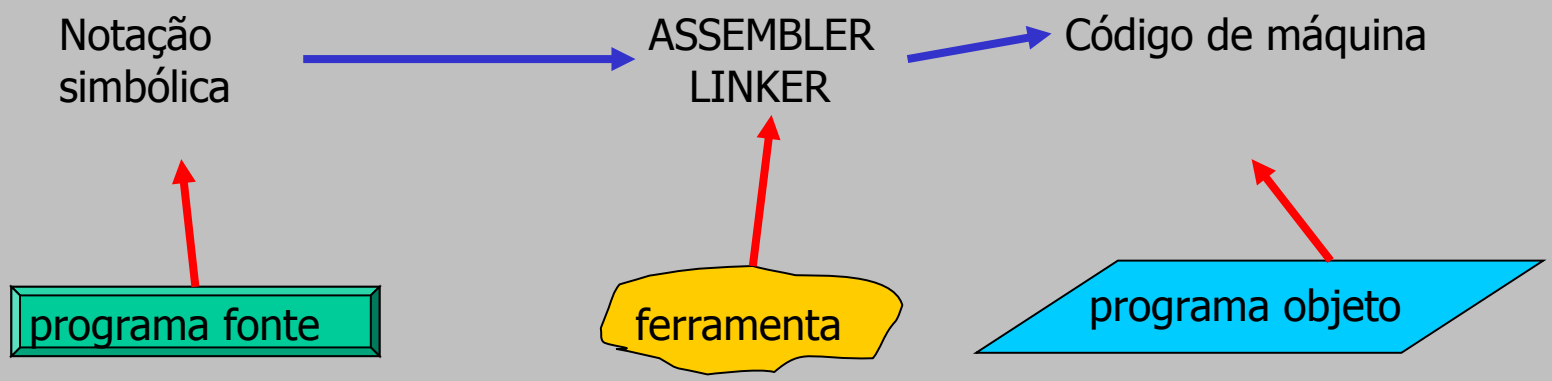


Fig. Fluxograma 2 – diagrama simplificado assemblagem

3.3 Linguagem de programação

Linguagem Assembly

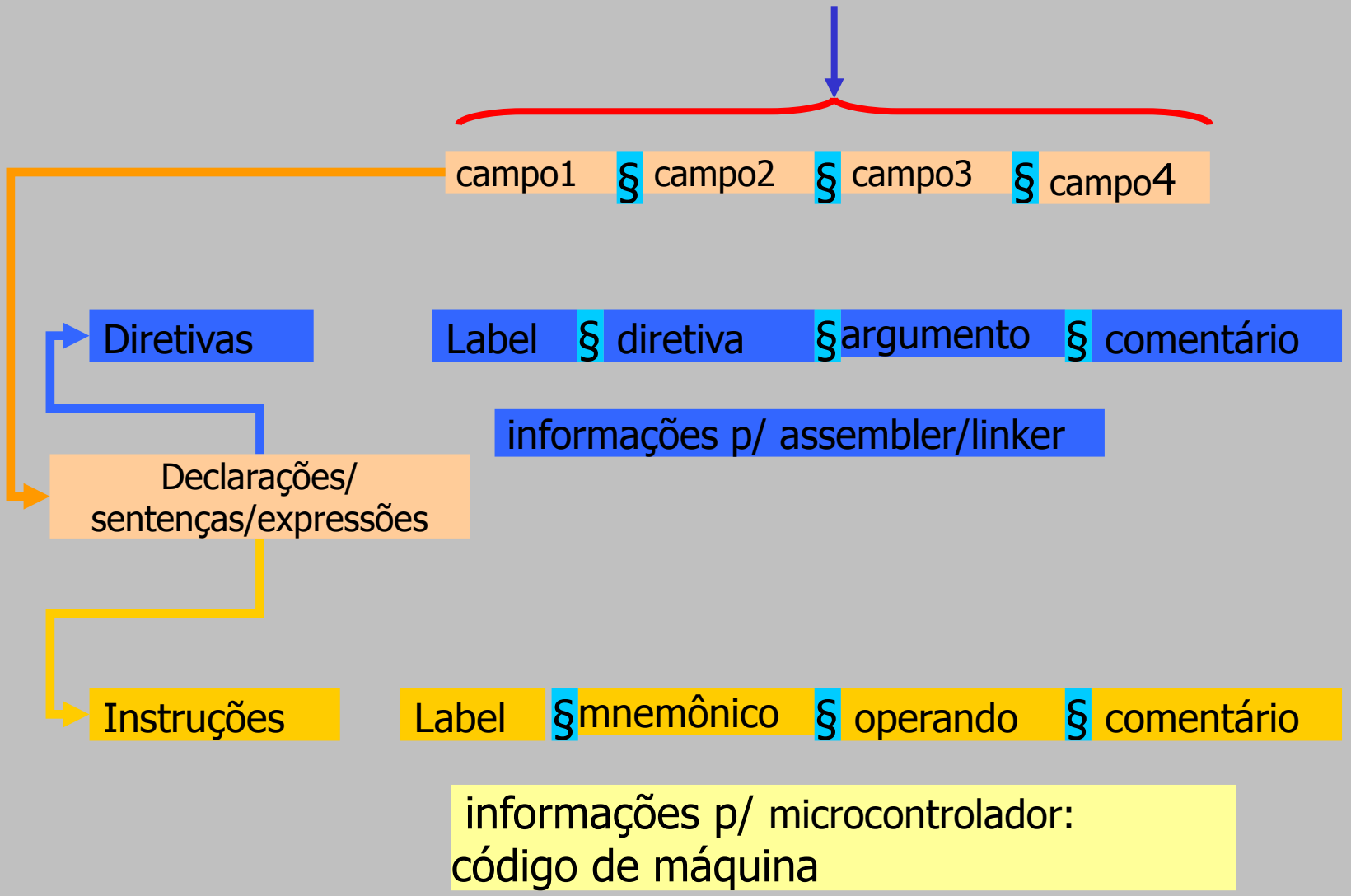


Linguagem Assembly

linguagem estruturada (rígida)

campo1 § campo2 § campo3 § campo4

3.4 Estrutura da linguagem Assembly



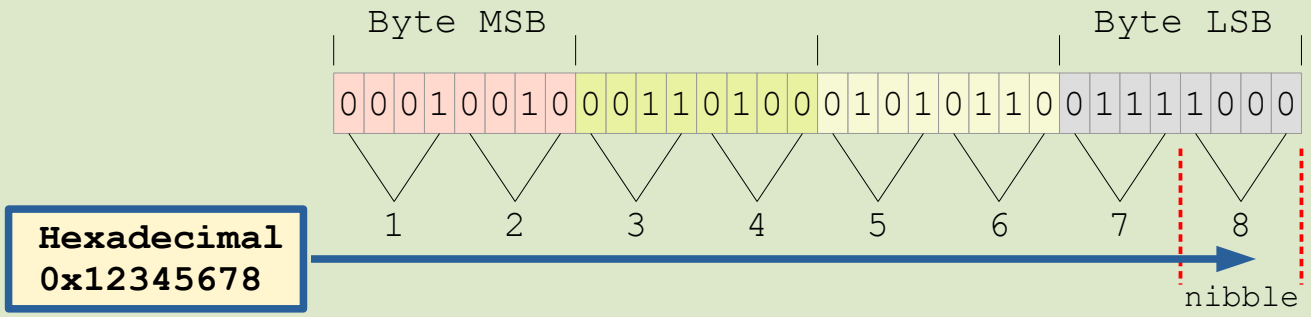
4. Software de programação de microcontroladores

µVision (Keil)

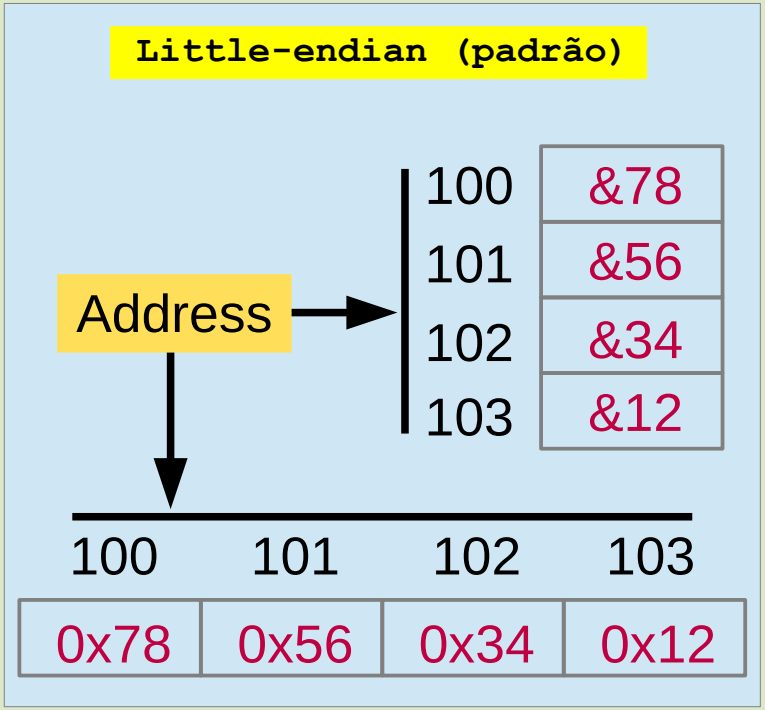
Siga as orientações do arquivo:
"Keil_uVision_instalacao.pdf"
e instale o programa uVision.

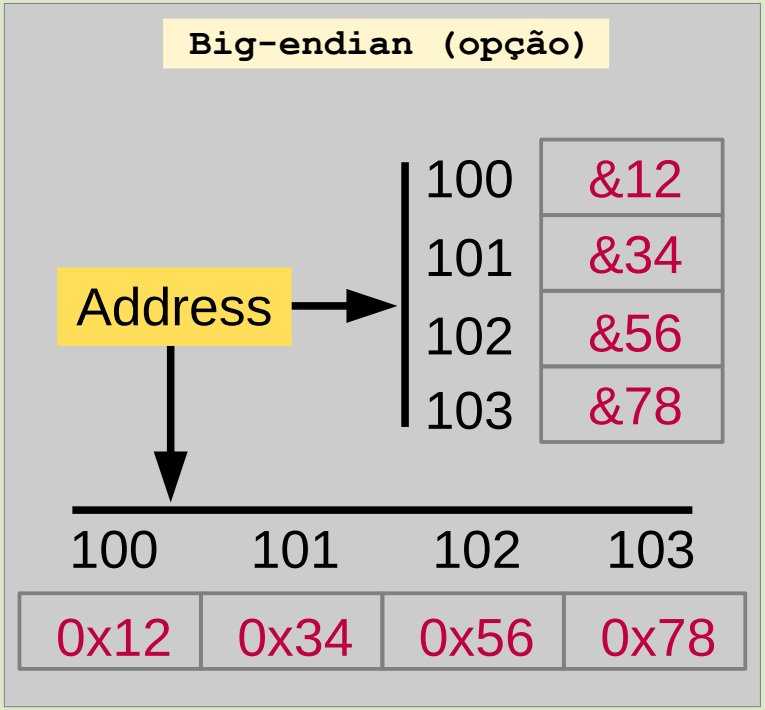
4.0 Programação de microcontroladores

ENDIANNESS



Registro µC





4.1 Programação de microcontroladores

USO DE DIRETIVAS

Os programas iniciais serão construído somente com o uso das diretivas (Assembler Directives). O objetivo é entender e assimilar o significado das DIRETIVAS em um programa de microcontrolador.

DIRETIVAS

EXPORT, AREA, END, DCB, DCW, DCD, EQU, SPACE, FILL

4.1.a Diretiva EXPORT

```
EXPORT {[WEAK]}

EXPORT symbol {[SIZE=n]}
```

symbol – é o nome (da variável, seção, procedimento, etc.) a ser exportado. A diferenciação entre letras maiúsculas e minúsculas se aplica a **symbol** .

A diretiva **EXPORT** possibilita que **symbol** seja usado por vários arquivos de um projeto.

Informações adicionais de todas as diretivas apresentadas devem ser consultadas no programa µVision, em HELP.

Ou

ARM Compiler v5.06 for µVision – version 5

Armasm User Guide

4.1.b Diretiva AREA

```
AREA sectionname{,attr}{,attr} …

          ou

AREA sectionname{,type}{,attr} …
```

A diretiva **AREA** informa ao Assembler para construir (assemblar) uma seção de código ou de dados. Na diretiva **AREA**:
sectionname – é o nome dado à seção. O nome **sectionname** faz diferenciação entre letras maiúsculas e minúsculas.
As palavras **attr** são os atributos que a diretiva **AREA** deve receber. Aqui, para facilitar o entendimento, o primeiro e o segundo **attr** serão, respectivamente, renomeados de: **type** e **attr**.
type – deve receber CODE ou DATA, o que indica o tipo de seção que será construída. Se CODE, o padrão (default) é READONLY; se DATA, o padrão (default) é READWRITE.
attr – pode (é opcional) receber READONLY se **type**=CODE *ou* READWRITE se **type**=DATA.

4.1.c Diretiva END

```
END

A diretiva END não tem argumentos
```

A diretiva **END** informa ao Assembler que ele atingiu o final do arquivo fonte.
Todo arquivo fonte, em linguagem Assembly, deve ser finalizado com a diretiva **END**.

4.1.d Diretiva DCB

```
{label}   DCB    expr1 {,expr2}
```

Em DCB, o B corresponde a um byte.

A diretiva **DCB** aloca (reserva) um ou mais bytes em memória e define o valor inicial para essa reserva de memória. O item **label** é opcional, porém pelo menos um de seus argumentos (**expr1**, **expr2**, ...) deve ser preenchido. O argumento da diretiva pode ser:
- um valor numérico passível de ser representado com 8 bits, na faixa de -128 a 127, ou 0 a 255;
- palavra ou frase entre aspas duplas; para um caracter qualquer use aspas simples.

4.1.d-1 PROGRAMA

1) Escreva um programa usando as diretivas EXPORT, AREA, DCB e END. A diretiva DCB é apropriada para uso de valores e caracteres. Use-a para várias possibilidades de valores em mais de uma base numérica e para caracteres ou frases. Assemblar e testar o programa; analisar o arquivo *.MAP e área de memória.

4.1.e Diretiva DCW

```
{label}   DCW    expr1 {,expr2}
```

Em DCW, o W representa uma ½ word (palavra), isto é, 2 bytes.

A diretiva **DCW** aloca (reserva) dois bytes (½ word) em memória e define o valor inicial para essa reserva de memória. O item **label** é opcional, porém pelo menos um de seus argumentos (**expr1**, **expr2**, ...) deve ser preenchido. O argumento da diretiva pode ser:
- um valor numérico passível de ser representado com 16 bits, na faixa de -32768 a 32767 ou 0 a 65535;

2) Refaça um novo programa usando a diretiva "DCW", para valores constantes (duplo byte ou half-word). Assemblar e testar o programa; analisar o arquivo *.MAP e área de memória.

4.1.f Diretiva DCD

```
{label}   DCD    expr1 {,expr2}
```

Em DCD, o D representa uma word (palavra), isto é, 4 bytes.

A diretiva **DCD** aloca (reserva) 4 bytes (uma word) em memória e define o valor inicial para essa reserva de memória. O item **label** é opcional, porém pelo menos um de seus argumentos (**expr1**, **expr2**, ...) deve ser preenchido. O argumento da diretiva pode ser:
- um valor numérico passível de ser representado com 32 bits.

3) Refaça um novo programa usando a diretiva "DCD", para valores constantes (4 bytes ou word). Assemblar e testar o programa; analisar o arquivo *.MAP e área de memória.

4.1.g Diretiva EQU

```
name   EQU    expr1 {,type}
```

A diretiva **EQU** atribui um valor numérico ao símbolo **name**, cujo uso é obrigatório.
O argumento **expr1** pode ser um valor numérico de até 32 bits, um endereço absoluto, etc.
O argumento **type** é opcional e pode ser: ARM, THUMB, etc.

4) Refaça o programa usando a diretiva "EQU". Assemblar e testar o programa; analisar o arquivo *.MAP e área de memória.

4.1.h Diretiva SPACE

```
{label}   SPACE expr
```

A diretiva **SPACE** reserva um ou mais espaços em memória e coloca nele(s) o valor zero.
O **label** é um rótulo opcional. O argumento **expr** define a quantidade bytes com valor nulo (zero).

5) Refaça o programa usando a diretiva "SPACE". Assemblar e testar o programa; analisar o arquivo *.MAP e área de memória.

4.1.i Diretiva FILL

```
{label}  FILL  expr{,value{,valuesize}}
```

A diretiva **FILL** reserva um ou mais espaços em memória e coloca nele(s) o valor **value**.
O rótulo **label** é opcional. O argumento **expr** define a quantidade bytes que receberá o valor **value**. O argumento **valuesize** indica, em número de bytes, o tamanho de **value**. O símbolo **valuesize** pode ser 1, 2 ou 4 (1 = 1 byte; 2 = 2 bytes ou ½ word; 4 = 4 bytes ou word).

6) Refaça o programa usando a diretiva "FILL". Assemblar e testar o programa; analisar o arquivo *.MAP e área de memória.