ELTD03z
Microcontroladores/Microprocessadores

Teoria_02a4

Prof. Enio R. Ribeiro

Universidade Federal de Itajubá - UNIFEI

1. Diagrama microcontrolador STM32F103 (completo)

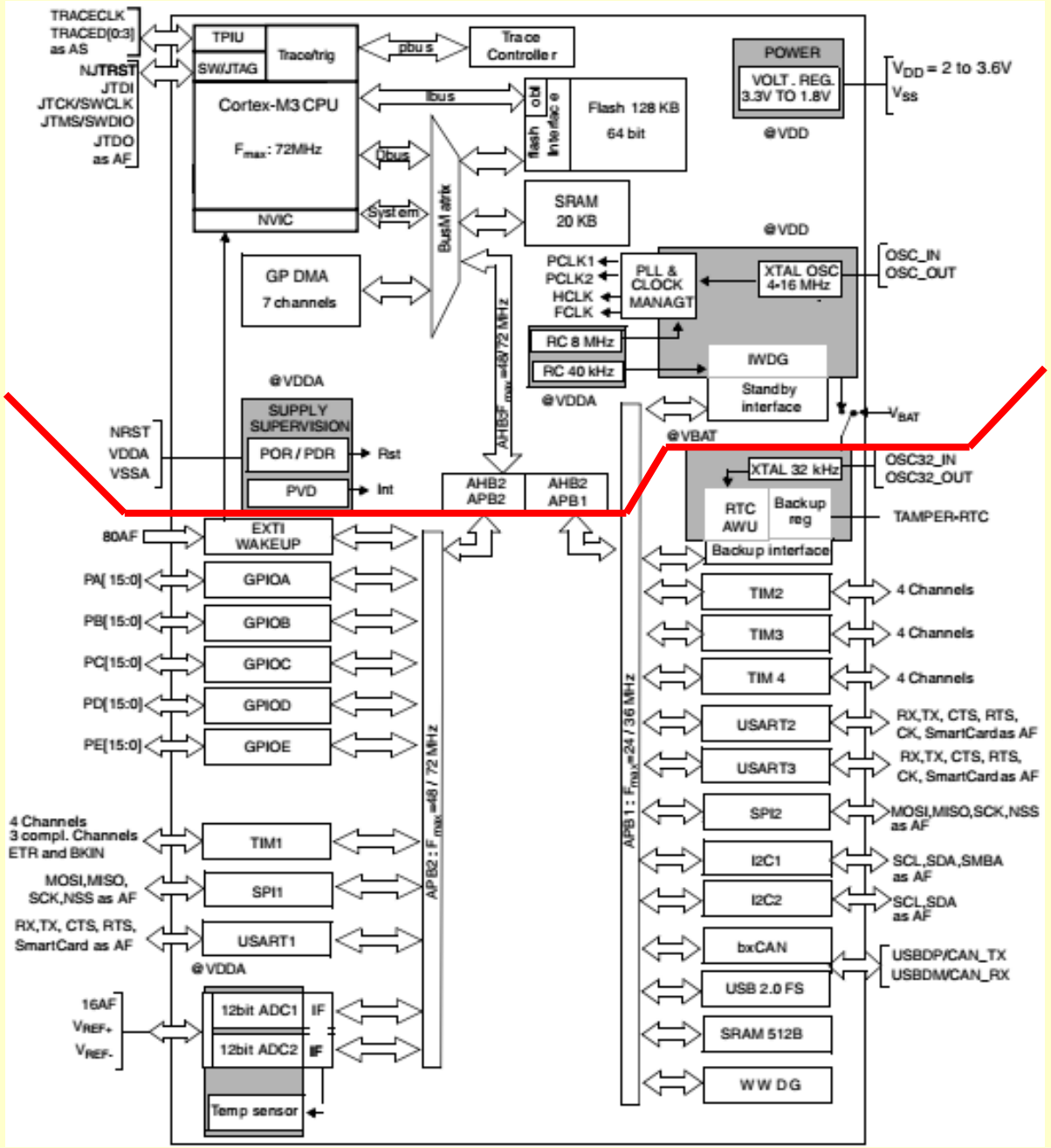


Fig. 1 – Diagrama em blocos: microcontrolador STM32F103 (stm32f103c8-1.pdf).

1. Diagrama microcontrolador STM32F103 (CPU+Mem.)

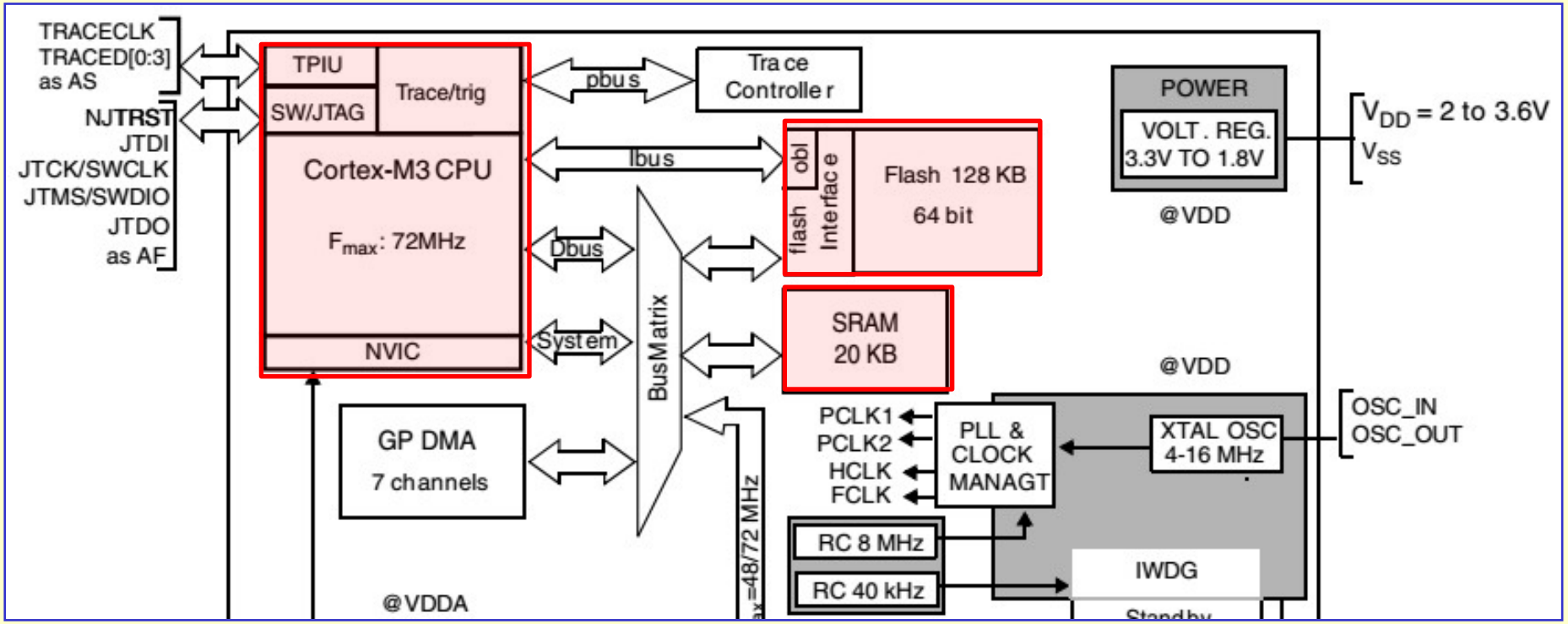


Fig. 2 – Diagrama em blocos: processador do STM32F103 (stm32f103c8-1.pdf).

1.1 Diagrama simplificado Cortex-M3 (CPU)

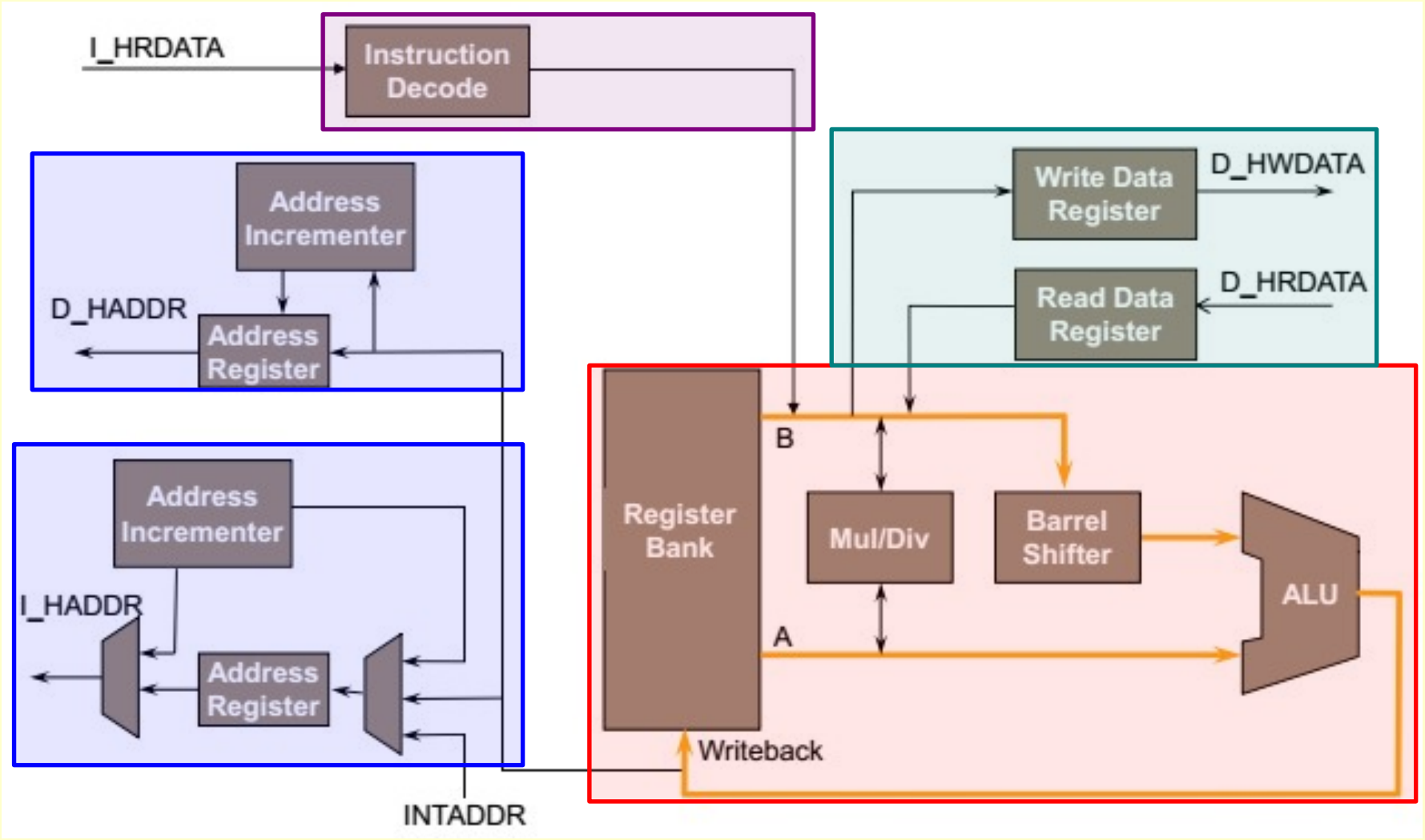
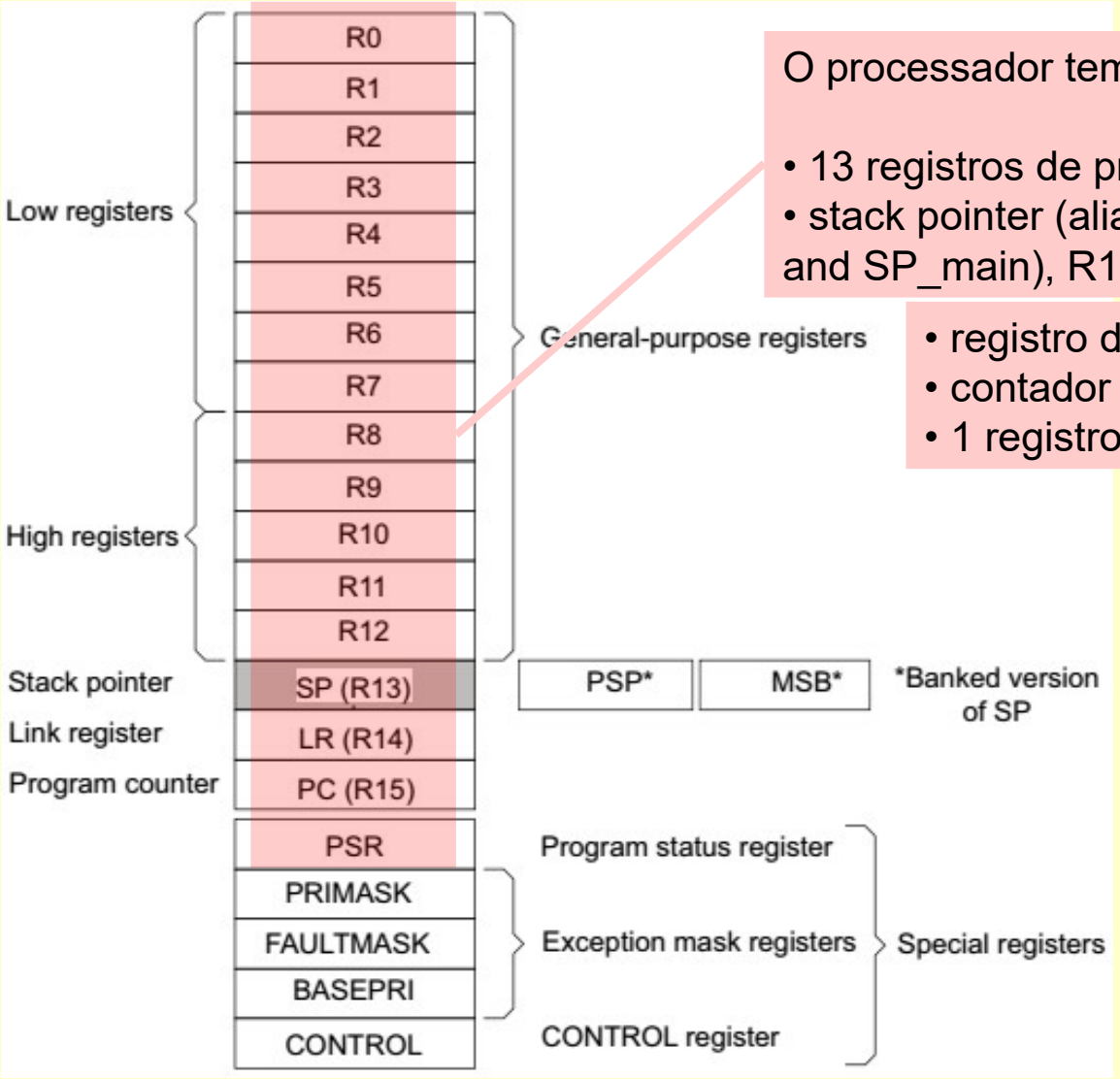


Fig. 3 – Diagrama simplificado Cortex-M3 core – [ARM_Architecture_Q3_11.pdf].

1.2 Modelo programável (funcional) CPU(STM32F1x)

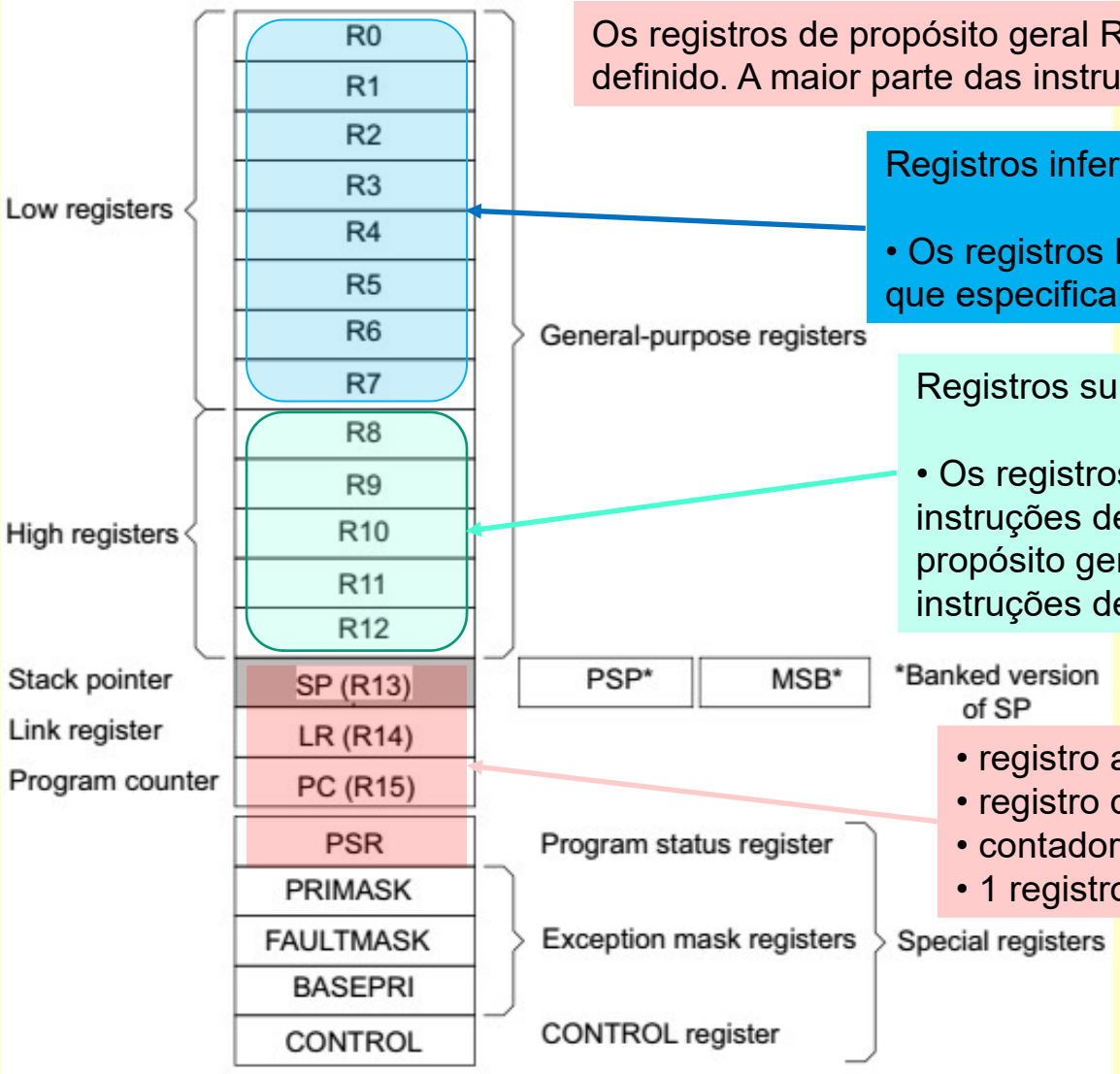


O processador tem os seguintes registros de 32 bits

- 13 registros de propósito geral, R0-R12
- stack pointer (alias of banked registers, SP_process and SP_main), R13
- registro de ligação, R14
- contador de programa, R15
- 1 registro de status de programa, PSR.

Fig. 4 – Modelo funcional da CPU microcontrolador STM32F1x (PM0056 Programming manual).

1.3 Registros da CPU(STM32F1x)



Os registros de propósito geral R0-R12 não têm uso arquitetural especial pré-definido. A maior parte das instruções podem acessar os registros R0-R12.

Registros inferiores (Low registers)

• Os registros R0-R7 são acessíveis por todas as instruções que especificam um registro de propósito geral.

Registros superiores (High registers)

• Os registros R8-R12 são acessíveis por todas as instruções de 32 bits que especificam um registro de propósito geral. Eles não são acessíveis por todas as instruções de 16 bits.

- registro apontador de pilha (Stack Pointer), R13
- registro de ligação (Link register), R14
- contador de programa (Program counter), R15
- 1 registro de status de programa, PSR.

2. Conjunto de instruções do STM32F103

STM32F103 => processador Cortex M3 => The processor implements the ARMv7-M Thumb instruction set.

STM32F103 => Thumb instructions are either 16-bit or 32-bit, and are aligned on a two-byte boundary. 16-bit and 32-bit instructions can be intermixed freely. Many common operations are most efficiently executed using 16-bit instructions.

HOWEVER:

- Most 16-bit instructions can only access eight of the general purpose registers, R0-R7 – (known as the low registers). A small number of 16-bit instructions can access the high registers, R8-R15;

- Many operations that would require two or more 16-bit instructions can be more efficiently executed with a single 32-bit instruction.

STM32F103

- tem 99 instruções;
- arquitetura ARM – é uma arquitetura de 32 bits;
- 8 bits => BYTE; 16 bits => HALFWORD; 32 bits => WORD.

2.1a Organização das instruções do STM32F103

STM32F103 instructions

- Memory Access Instructions;
- Data Processing Instructions;
- Saturating Instructions;
- Bitfield Instructions;
- Branch and Control Instructions;
- Miscellaneous Instructions.

2.1.1 Memory access instructions

Mnemonic	Brief description	Section
ADR	Load PC-relative address	*ADR on page 60*
CLREX	Clear exclusive	*CLREX on page 71*
LDM{mode}	Load multiple registers	*LDM and STM on page 67*
LDR{type}	Load register using immediate offset	*LDR and STR, immediate offset on page 61*
LDR{type}	Load register using register offset	*LDR and STR, register offset on page 63*
LDR{type}T	Load register with unprivileged access	*LDR and STR, unprivileged on page 64*
LDR	Load register using PC-relative address	*LDR, PC-relative on page 65*
LDREX{type}	Load register exclusive	*LDREX and STREX on page 70*
POP	Pop registers from stack	*PUSH and POP on page 68*
PUSH	Push registers onto stack	*PUSH and POP on page 68*
STM{mode}	Store multiple registers	*LDM and STM on page 67*
STR{type}	Store register using immediate offset	*LDR and STR, immediate offset on page 61*
STR{type}	Store register using register offset	*LDR and STR, register offset on page 63*
STR{type}T	Store register with unprivileged access	*LDR and STR, unprivileged on page 64*
STREX{type}	Store register exclusive	*LDREX and STREX on page 70*

Tabela 24 – Instruções de acesso a memória – fonte todas tabelas (PM0056 Programming manual).

2.1.2 Data processing instructions

Mnemonic	Brief description	See
ADC	Add with carry	*ADD, ADC, SUB, SBC, and RSB on page 73*
ADD	Add	*ADD, ADC, SUB, SBC, and RSB on page 73*
ADDW	Add	*ADD, ADC, SUB, SBC, and RSB on page 73*
AND	Logical AND	*AND, ORR, EOR, BIC, and ORN on page 75*
ASR	Arithmetic shift right	*ASR, LSL, LSR, ROR, and RRX on page 76*
BIC	Bit clear	*AND, ORR, EOR, BIC, and ORN on page 75*
CLZ	Count leading zeros	*CLZ on page 77*
CMN	Compare negative	*CMP and CMN on page 78*
CMP	Compare	*CMP and CMN on page 78*
EOR	Exclusive OR	*AND, ORR, EOR, BIC, and ORN on page 75*
LSL	Logical shift left	*ASR, LSL, LSR, ROR, and RRX on page 76*
LSR	Logical shift right	*ASR, LSL, LSR, ROR, and RRX on page 76*
MOV	Move	*MOV and MVN on page 79*

Tabela 27 – Instruções de processamento de dados (PM0056 Programming manual).

2.1.2 Data processing instructions

Mnemonic	Brief description	See
MOVT	Move top	*MOVT on page 80*
MOVW	Move 16-bit constant	*MOV and MVN on page 79*
MVN	Move NOT	*MOV and MVN on page 79*
ORN	Logical OR NOT	*AND, ORR, EOR, BIC, and ORN on page 75*
ORR	Logical OR	*AND, ORR, EOR, BIC, and ORN on page 75*
RBIT	Reverse bits	*REV, REV16, REVSH, and RBIT on page 81*
REV	Reverse byte order in a word	*REV, REV16, REVSH, and RBIT on page 81*
REV16	Reverse byte order in each halfword	*REV, REV16, REVSH, and RBIT on page 81*
REVSH	Reverse byte order in bottom halfword and sign extend	*REV, REV16, REVSH, and RBIT on page 81*
ROR	Rotate right	*ASR, LSL, LSR, ROR, and RRX on page 76*
RRX	Rotate right with extend	*ASR, LSL, LSR, ROR, and RRX on page 76*
RSB	Reverse subtract	*ADD, ADC, SUB, SBC, and RSB on page 73*
SBC	Subtract with carry	*ADD, ADC, SUB, SBC, and RSB on page 73*
SUB	Subtract	*ADD, ADC, SUB, SBC, and RSB on page 73*
SUBW	Subtract	*ADD, ADC, SUB, SBC, and RSB on page 73*
TEQ	Test equivalence	*TST and TEQ on page 82*
TST	Test	*TST and TEQ on page 82*

Tabela 27 – Instruções de processamento de dados (PM0056 Programming manual).

2.1.3 Multiply and divide instructions

Mnemonic	Brief description	See
MLA	Multiply with accumulate, 32-bit result	*MUL, MLA, and MLS on page 83*
MLS	Multiply and subtract, 32-bit result	*MUL, MLA, and MLS on page 83*
MUL	Multiply, 32-bit result	*MUL, MLA, and MLS on page 83*
SDIV	Signed divide	*SDIV and UDIV on page 86*
SMLAL	Signed multiply with accumulate (32x32+64), 64-bit result	*UMULL, UMLAL, SMULL, and SMLAL on page 85*
SMULL	Signed multiply (32x32), 64-bit result	*UMULL, UMLAL, SMULL, and SMLAL on page 85*
UDIV	Unsigned divide	*SDIV and UDIV on page 86*
UMLAL	Unsigned multiply with accumulate (32x32+64), 64-bit result	*UMULL, UMLAL, SMULL, and SMLAL on page 85*
UMULL	Unsigned multiply (32x32), 64-bit result	*UMULL, UMLAL, SMULL, and SMLAL on page 85*

Tabela 28 – Instruções de multiplicação e divisão (PM0056 Programming manual).

SSAT and USAT

Signed saturate and unsigned saturate to any bit position, with optional shift before saturating.

2.1.5 Bitfield instructions

A Tabela 29 mostra as instruções que operam em conjunto de bits adjacentes em registros e em campos de bits.

Mnemonic	Brief description	See
BFC	Bit field clear	*BFC and BFI on page 89*
BFI	Bit field insert	*BFC and BFI on page 89*
SBFX	Signed bit field extract	*SBFX and UBFX on page 89*
SXTB	Sign extend a byte	*SXT and UXT on page 90*
SXTH	Sign extend a halfword	*SXT and UXT on page 90*
UBFX	Unsigned bit field extract	*SBFX and UBFX on page 89*
UXTB	Zero extend a byte	*SXT and UXT on page 90*
UXTH	Zero extend a halfword	*SXT and UXT on page 90*

Tabela 29 – Packing and unpacking instructions (PM0056 Programming manual).

2.1.6 Branch and control instructions

Mnemonic	Brief description	See
B	Branch	*B, BL, BX, and BLX on page 92*
BL	Branch with Link	*B, BL, BX, and BLX on page 92*
BLX	Branch indirect with Link	*B, BL, BX, and BLX on page 92*
BX	Branch indirect	*B, BL, BX, and BLX on page 92*
CBNZ	Compare and Branch if Non Zero	*CBZ and CBNZ on page 93*
CBZ	Compare and Branch if Non Zero	*CBZ and CBNZ on page 93*
IT	If-Then	*IT on page 94*
TBB	Table Branch Byte	*TBB and TBH on page 96*
TBH	Table Branch Halfword	*TBB and TBH on page 96*

Tabela 30 – Instruções de desvio e controle (PM0056 Programming manual).

2.1.7 Miscellaneous instructions

Mnemonic	Brief description	See
BKPT	Breakpoint	*BKPT on page 98*
CPSID	Change Processor State, Disable Interrupts	*CPS on page 98*
CPSIE	Change Processor State, Enable Interrupts	*CPS on page 98*
DMB	Data Memory Barrier	*DMB on page 99*
DSB	Data Synchronization Barrier	*DSB on page 100*
ISB	Instruction Synchronization Barrier	*ISB on page 100*
MRS	Move from special register to register	*MRS on page 100*
MSR	Move from register to special register	*MSR on page 101*
NOP	No Operation	*NOP on page 102*
SEV	Send Event	*SEV on page 102*
SVC	Supervisor Call	*SVC on page 103*
WFE	Wait For Event	*WFE on page 103*
WFI	Wait For Interrupt	*WFI on page 104*

Tabela 32 – Instruções diversas (PM0056 Programming manual). 13

3 Modos de endereçamento STM32F1x

- IMEDIATO (IMMEDIATE) – IMM;

- INERENTE (INHERENT) – USE OF REGISTERS;

3.1 Modos de endereçamento: IMEDIATO

Imediato (Immediate - IMM)

The operand of the instruction is the “actual data” to be loaded into a register.

The data itself is contained in the instruction.
The “#” sign indicates to the assembler that “immediate addressing” is intended.

Instruction: **MOV**

The MOV instruction copies the value of *Operand2* into *Rd*.
Syntax

```
MOV Rd, Operand2 ;Operand2=#imm16
```

- ‘*Rd*’ is the destination register
- ‘*imm16*’ is any value in the range 0—65535

Example:

```
MOV R1, #0xAFE ;R1[31:0]=#imm16
```

3.1 Modos de endereçamento: IMEDIATO

Instruction: **MOVW**

The MOVW instruction copies the value of *Operand2* into *Rd*.

Syntax

```
MOVW Rd, Operand2 ;Operand2=#imm16
```

- '*Rd*' is the destination register
- '*imm16*' is any value in the range 0—65535

Example:

```
MOVW R2, #0x19EC  ;R2[31:0]=#imm16
```

3.1 Modos de endereçamento: IMEDIATO

Instruction: **MOVT**

The MOVT instruction writes a 16-bit immediate value *imm16*, to the top halfword, *Rd*[31:16], of its destination register. **The write does not affect *Rd*[15:0].**

Syntax

```
MOVT Rd, Operand2 ;Operand2=#imm16
```

- '*Rd*' is the destination register
- '*imm16*' is any value in the range 0—65535

Example:

```
MOVT R3, #0xEC2    ;R3[31:16]=#imm16
```

3.1 Modos de ender.: imediato (exercícios)

Ex_1a: Escreva um programa para atribuir valor de 1 byte em dois registradores. Use a instrução ***mov*** com endereçamento imediato e as bases numéricas: decimal e hexadecimal. Assemblar e testar o programa.

Ex_1b: Escreva um programa para atribuir valor de 1 byte em dois registradores. Use a instrução ***mov*** com endereçamento imediato e as bases numéricas: binária e octal. Assemblar e testar o programa.

Ex_2a: Escreva um programa para atribuir valor de 1 byte em um registrador. Use a instrução ***mov*** com endereçamento imediato e as bases numéricas: decimal ou hexadecimal. Use a diretiva ***equ*** para representar o valor imediato. Assemblar e testar o programa.

Ex_2b: Escreva um programa para atribuir valor de 1 byte em um registrador. Use a instrução ***mov*** com endereçamento imediato e as bases numéricas: binária e octal. Use a diretiva ***equ*** para representar o valor imediato. Assemblar e testar o programa.

Ex_3a: Escreva um programa para atribuir valor de 2 bytes em 1 registrador. Use a instrução ***movw*** com endereçamento imediato e a base numérica: hexadecimal. Assemblar e testar o programa.

Ex_3b: Escreva um programa para atribuir valor de 2 bytes em 3 registradores. Use a instrução ***movw*** com endereçamento imediato e as bases numéricas: binária, octal e decimal. Use a diretiva ***equ*** para representar os valores imediatos. Assemblar e testar o programa.

Ex_4a: Escreva um programa para atribuir valor de 2 bytes em 1 registrador. Use a instrução ***movt*** com endereçamento imediato e a base numérica: hexadecimal. Assemblar e testar o programa.

Ex_4b: Escreva um programa para atribuir valor de 2 bytes em 3 registradores. Use a instrução ***movt*** com endereçamento imediato e as bases numéricas: binaria, octal e decimal. Assemblar e testar o programa.

Ex_4c: Escreva um programa para atribuir valor de 2 bytes em 3 registradores. Use a instrução ***movt*** com endereçamento imediato e as bases numéricas: binaria, octal e decimal. Use a diretiva ***equ*** para representar os valores imediatos. Assemblar e testar o programa.

Ex_4d: Escreva um programa para atribuir valor de 4 bytes em 3 registradores. Use a instrução ***mov e movt*** com endereçamento imediato e as bases numéricas: binaria, octal e decimal. Use a diretiva ***equ*** para representar os valores imediatos. Assemblar e testar o programa.

3.1 Modos de endereçamento: IMEDIATO

Instruction: **ADD**

The ADD instruction adds the immediate (imm12) value to the value in register Rn, and writes the result to the destination register.

Syntax

```
ADD Rd,Rn,Operand2 ;(Operand2)=(#imm12)
```

Instruction: **SUB**

The SUB instruction subtracts an immediate (imm12) value from the value in Rn, and writes the result to the destination register Rd.

Syntax

```
SUB Rd,Rn,Operand2 ;(Operand2)=(#imm12)
```

For both instructions: **ADD/SUB**

- '*Rd*' is the destination register. If '*Rd*' is omitted, the destination register is '*Rn*'
- '*Rn*' is the register holding the first operand
- '*imm12*' is any value in the range 0—4095

3.1 Modos de endereçamento: IMEDIATO

Example: **ADD**

```
ADD R0,R1,#0x800 ;R0=R1+0x800
```

Example: **SUB**

```
SUB R1,R2,#0x7FF ;R1=R2-0x7FF
```

3.1 Modos de ender.: imediato (exercícios)

Ex_5a: Escreva um programa para adicionar valor de 1 byte em 1 registrador. Use a instrução ***add*** com endereçamento imediato e as bases numéricas: decimal ou hexadecimal. Assemblar e testar o programa.

Ex_5b: Escreva um programa para adicionar valor de 1 byte em 1 registrador. Use a instrução ***add*** com endereçamento imediato e as bases numéricas: binária, decimal e hexadecimal. Assemblar e testar o programa.

Ex_5c: Escreva um programa para adicionar valor de 1 byte em 1 registrador. Use a instrução ***add*** com endereçamento imediato e as bases numéricas: binária, decimal e hexadecimal. Use a diretiva ***equ*** para representar o valor imediato. Assemblar e testar o programa.

Ex_5d: Escreva um programa para adicionar 3 valores de 1 byte em 1 registrador. Use a instrução ***add*** com endereçamento imediato e as bases numéricas: binária, decimal e hexadecimal. Use a diretiva ***equ*** para representar o valor imediato. Assemblar e testar o programa.

Ex_5e: Escreva um programa para fazer a subtração de: vc1 – vc2 (onde 255<vc1<65535 e 0<vc2<256). Use a instrução ***sub*** com endereçamento imediato e as bases numéricas: binária, decimal e hexadecimal. Use a diretiva ***equ*** para representar os valores imediatos. Assemblar e testar o programa.

3.2 Modos de endereçamento: INERENTE (REGISTER)

Inerente (Inherent - register)

The operand of the instruction is the content of a *source register* to be copied into a *destination register*.

Instruction: **MOV**

The MOV instruction copies the value of *operand2* into *Rd*.

Syntax

```
MOV Rd, Operand2 ;
```

- '*Rd*' is the destination register
- '*Operand2*' - *Operand2* is a flexible second operand and, for an inherent addressing mode, it is a: *Register (without optional shift)*

Example:

```
MOV R1, R2   ;R1=R2
```

3.2 Modos de endereçamento: INERENTE (REGISTER)

Instruction: **ADD**

The ADD (register) instruction adds a register value and an optionally-shifted register value, and writes the result to the destination register.

Syntax

```
ADD {<Rd>}, <Rn> , Operand2
```

Instruction: **SUB**

The SUB (register) instruction subtracts an optionally-shifted register value from a register value, and writes the result to the destination register.

Syntax

```
SUB {<Rd>}, <Rn>, Operand2
```

For both instructions: **ADD/SUB**

- '<*Rd*>' Specifies the destination register. If <*Rd*> is omitted, this register is the same as <*Rn*>
- '<Rn>' Specifies the register that contains the first operand
- 'Operand2' is a register specified in the form *Rm {,shift}*. Rm is the register holding the data for the second operand. **(At the moment: disregard the {,shift})**

3.2 Modos de endereçamento: INERENTE (REGISTER)

Example: **ADD**

```
ADD {<Rd>}, <Rn>, Operand2
ADD R0, R1, R2      ;R0=R1+R2
```

Example: **SUB**

```
SUB {<Rd>}, <Rn>, Operand2
SUB R1,R2,R0    ;R1=R2-R0
```

3.2 Modos de endereçamento: INERENTE (REGISTER)

Ex_6a: Escreva um programa para copiar o valor de 1 registrador em outro (valor inicial: de 1 byte). Use a instrução ***mov*** com endereçamento inerente. Use a diretiva equate (equ) para o valor inicial. Assemblar e testar o programa.

Ex_6b: Repita o exercício “Ex_6a” considerando um valor de 2 bytes, na metade menos significativa, no registrador inicial.

Ex_6c: Repita o exercício “Ex_6a” considerando um valor de 2 bytes, na metade mais significativa, do registrador inicial.

Ex_7a: Escreva um programa para copiar o valor de 1 registrador em outro (valor inicial: de 1 byte). Use a instrução ***mov*** e após faça uma operação adição entre eles. Coloque o resultado em dos registradores usado inicialmente. Use endereçamento inerente. Use a diretiva equate (equ) para o valor inicial. Assemblar e testar o programa.

Ex_7b: Escreva um programa para copiar o valor de 1 registrador em dois outros registradores (valor inicial: de 1 byte). Use a instrução ***mov*** e após faça uma operação adição entre eles. Coloque o resultado em dos registradores usado inicialmente. Use endereçamento inerente. Use a diretiva equate (equ) para o valor inicial. Assemblar e testar o programa.

Ex_7c: Repita o exercício “Ex_7b” considerando o valor inicial de 2 bytes