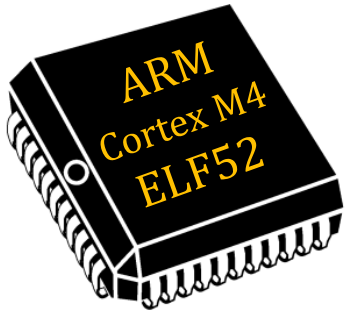


UTFPR

- Linguagem
- Assembly
- Arquitetura
- Cortex-M
- Cortex-M-ISA
- Referências

Microcontroladores: (LT36D)
Prof: DaLuz

UTFPR

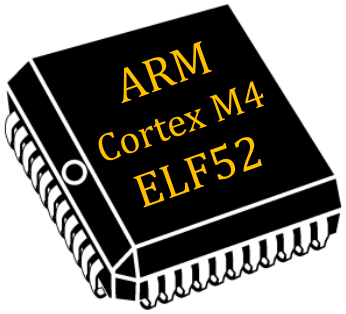
Linguagem

- Um microcontrolador **executa** alguns comandos específicos, que são lidos por ele através de códigos binários.
- Estes comandos ou ***object codes*** constituem a linguagem de máquina.
- As instruções escritas em **assembly** ou os comandos de uma **linguagem alto nível**, são traduzidos em linguagem de máquina.

UTFPR

Linguagem

- A linguagem **Assembly**, possui um conjunto de instruções, que são conhecidos como símbolos (**mnemônicos**) ou (**opcodes**).
- A conversão da linguagem **Assembly** para a linguagem de máquina é feita pelo **assembler** (montador equivalente ao compilador).
- Entretanto esta linguagem não é universal. Ela é **específica** e cada **CPU** tem a sua, sendo considerada uma linguagem de baixo nível.



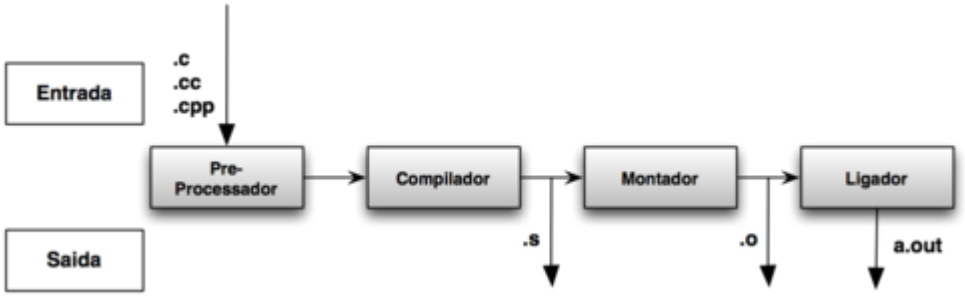
UTFPR

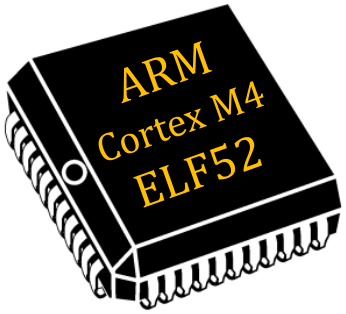
Linguagem

- As linguagens de **alto nível** são aquelas mais próximas do entendimento **humano**.

  Ex.: **C**, **C++**, Java, Pascal, etc ...

- Estas linguagens são **compiladas** para gerar o **código** de máquina.





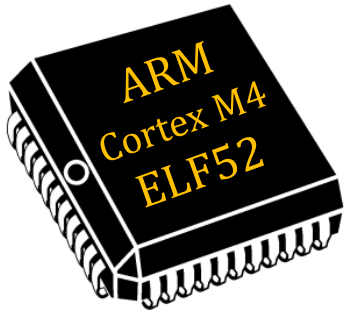
UTFPR

Assembly

- ARM **Cortex-M4** utiliza uma tecnologia conhecida como **Thumb2**.

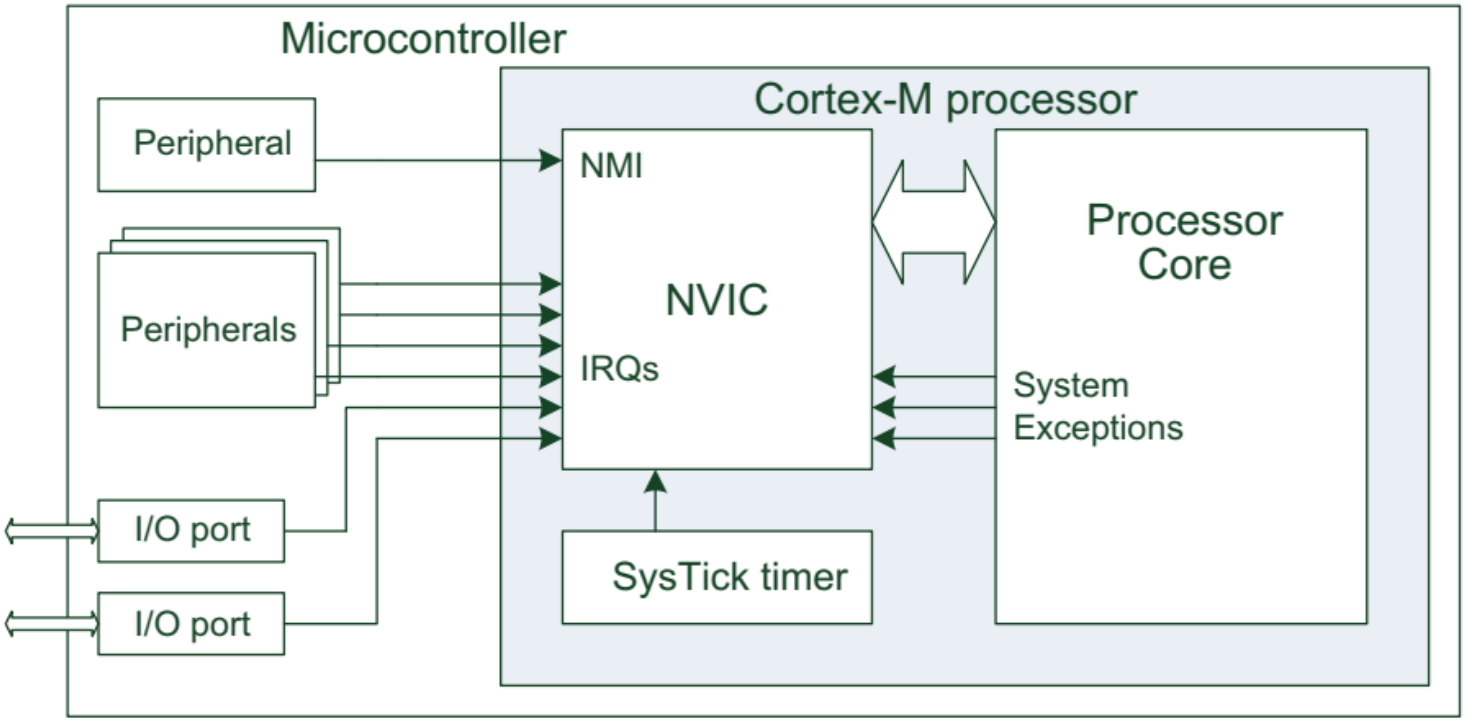
  - Instruções de **16-bit** (Thumb) e **32-bit** (ARM)
  - **Thumb2** é uma junção de (ARM+Thumb).

- Arquitetura **RISC** com arquitetura **LOAD/STORE**.

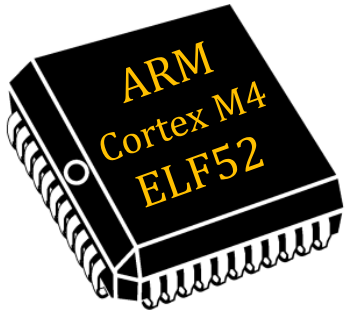


Arquitetura

Cortex-M



*Ref. **



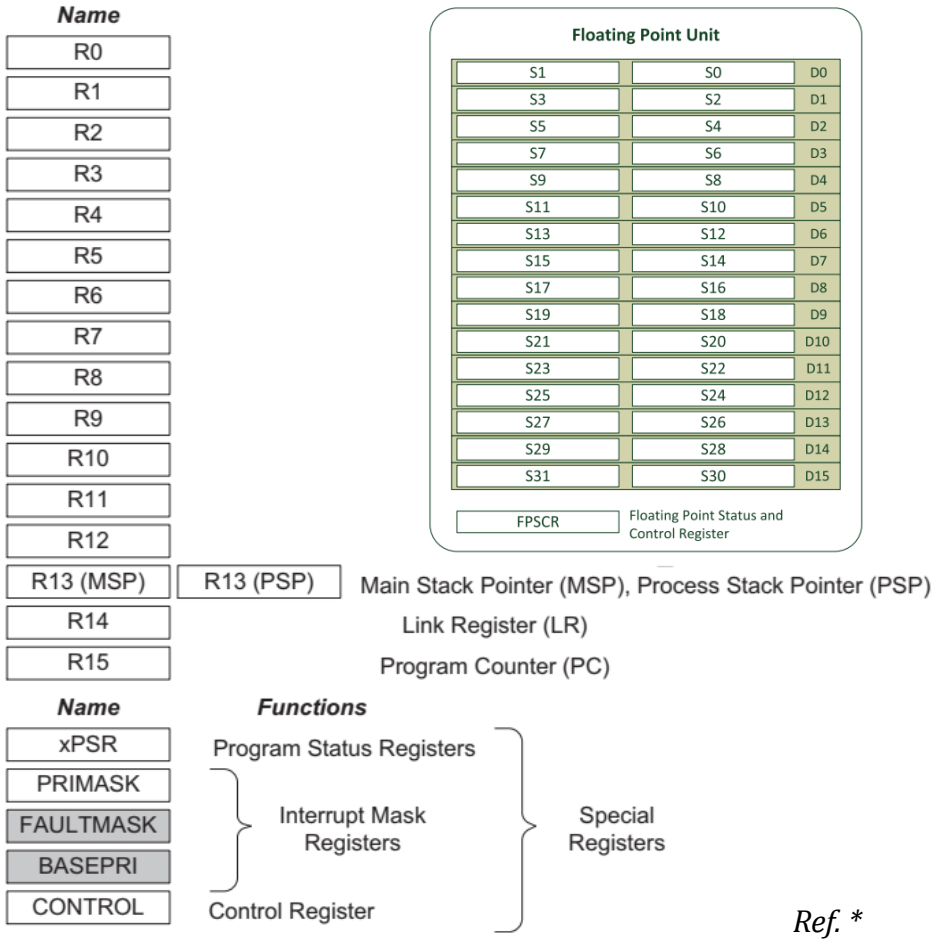
Cortex-M

Registradores

- 1) Geral (R0-R15)
- 2) F.P. (S0-S31)
- 3) S.R. (xPSR, Primask, Faultmask, Basepri, Control)

- *Low registers: R0-R7*
- *High registers: R8-R15*
- *Special: SP, LR, PC*

- *Obs: Todos 32-bit*



*Ref. **

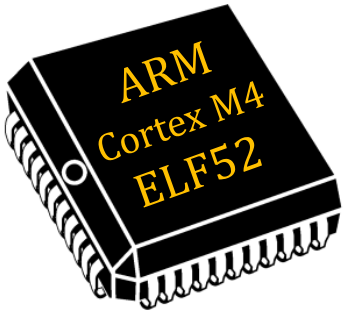
UTFPR

Cortex-M

Tipos de Dados

Instruções do núcleo Cortex-M (**Revisão**):

- **bit:** guarda apenas um bit (**0** ou **1**).
- **byte:** conjunto de **8-bit** = 1 x endereço **RAM**
- **half-word:** **16-bit** = 2 x endereço **RAM**
- **word:** **32-bit** = 4 x endereço **RAM**
- **double-word:** **64-bit** = ex. R1:R0 = 8 x endereço **RAM**



UTFPR

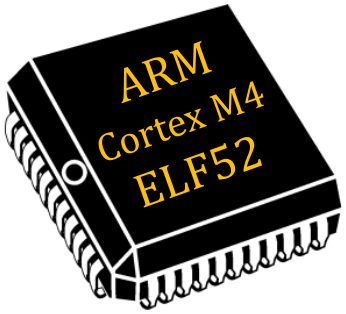
Cortex-M

Registradores

Registradores: R0-R12, R13(SP), R14(LR), R15(PC)															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Uso Geral								Uso Geral							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Uso Geral								Uso Geral							

- Registradores R0 a R15 – R0 a R12 ⇨ 13 de uso geral;
- Acessíveis para a maioria das instruções L:R0-R7, H:R8-R15
- **b0** é o *Least Significant bit* (**LSb**) e **b31** é o *Most* S. b. (**MSb**)
- *Capacidade:* **32-bit** ⇨ 8 dígitos hexad. ⇨ 0x1A2B.3C4D

  0 a *4.294.967.295 (U) ou -2,147,483,648 a 2.147.384.647 (S)*
- *Os valores podem representar: números ou endereços.*



UTFPR

Cortex-M

xPSR = APSR + EPSR + IPSR

APSR - *Application Program Status Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	Reservado							GE[3:0]				Reservado															

EPSR - *Execution Program Status Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reservado					ICI/IT		T	Reservado								ICI/IT						Reservado									

IPSR - *Interrupt Program Status Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reservado																							0 ou Número da Exceção								

xPSR - Acesso Combinado - *Application Program Status Register*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	Z	C	V	Q	ICI/IT		T	*Reservado*				GE[3:0]				ICI/IT						*R.*	0 ou Número da Exceção								

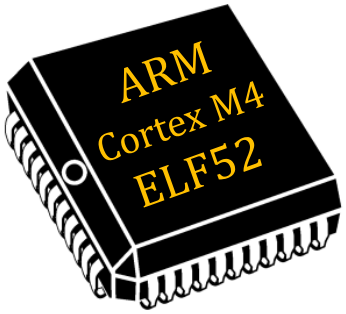
- Os três registradores **APSR**, **EPSR** e **IPSR** podem ser acessados individualmente ou combinados no **xPSR** ou dois a dois **IAPSR**, **EAPSR ou IEPSR.**

UTFPR

Cortex-M

Campos dos bits xPSR

Bit Fields - Campo de bits no *program status register*	
N	Negative flag
Z	Zero flag
C	Carry (or NOT borrow) flag
V	Overflow flag
Q	Sticky saturation flag (not available ARMv6-M)-**DSP**
GE[3:0]	Greater-Than or Equal flags for each byte lane (ARMv7E-M only; not available in ARMv6-M or Cortex-M3)-**SIMD**.
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit for conditional execution (not available in ARMv6-M).
T	Thumb state, always **1**(Cortex-**M**); trying to clear this bit will cause a fault exception. (0 or 1) in Cortex-R or Cortex-A
Exception Number	Indicates which exception the processor is handling.



UTFPR

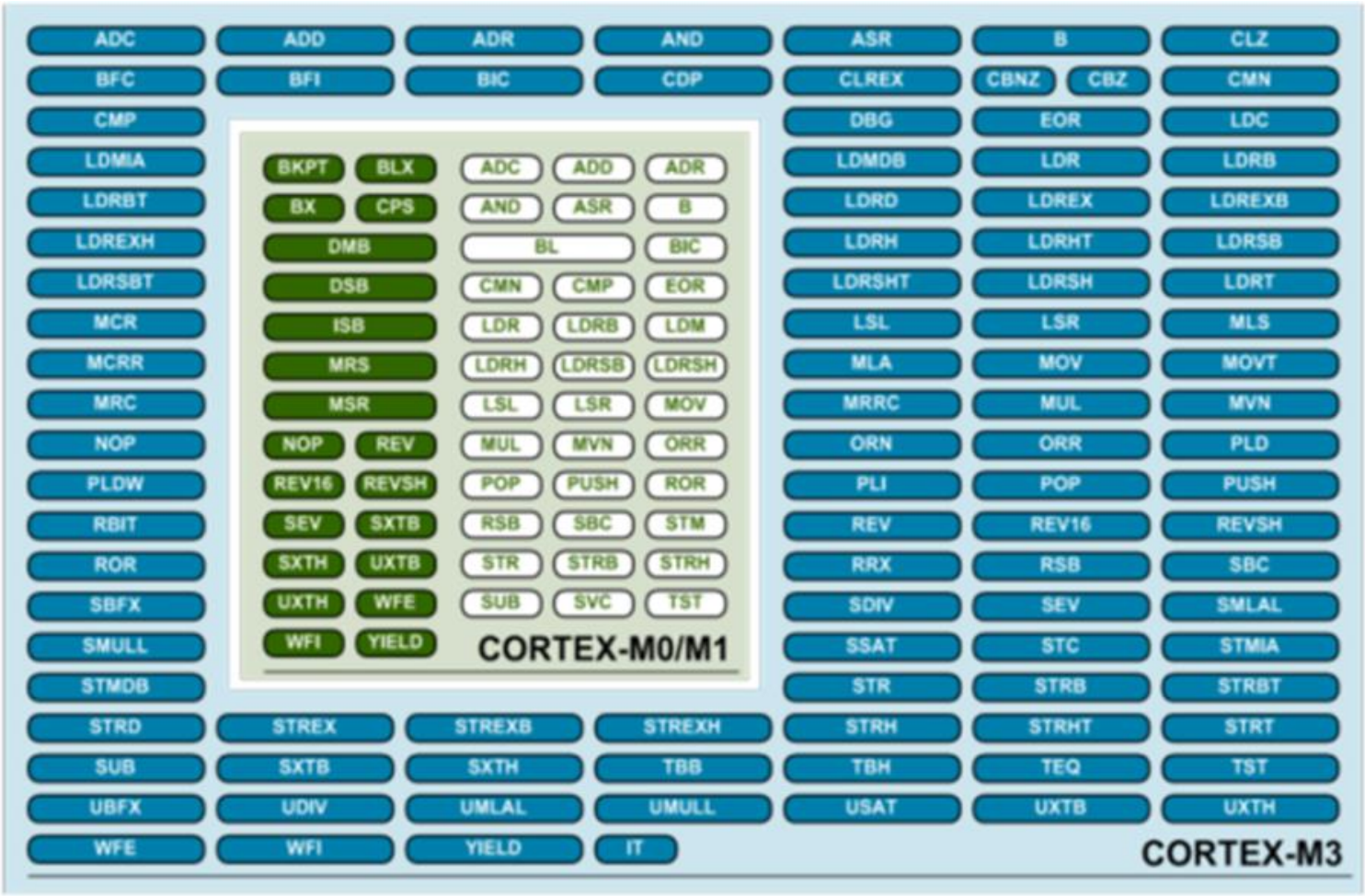
Cortex-M

Cortex-M *Instruction Set Architecture*

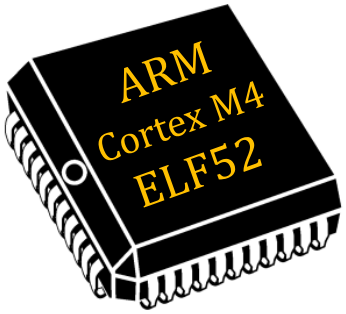
Texas ISA (221pgs):
http://www.elf52.daeln.com.br/Pdfs/CortexM_InstructionSet.pdf

ARM ISA Cap.11:
http://www.elf52.daeln.com.br/Pdfs/Arm_Asm_User_Guide_v5.06.pdf

Cortex-M ISA



*Ref. **



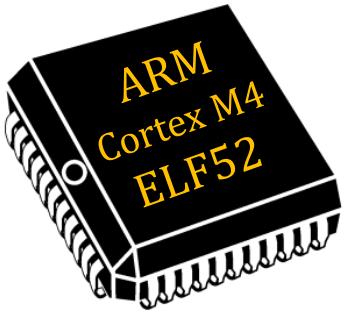
UTFPR

Cortex-M ISA

Instruções

Instruções do núcleo Cortex-M:

- **Tipo de dados comuns**
- **Modos de operação**
- **Registradores**
- **Instruções**
- **Acesso a Memória**
- **Exceções / Interrupções / *H. Faults***



UTFPR

Cortex-M ISA

Instruções: Genérica

Label

```
MNEMÔNICO Destino, Operando1,                ;Comentário
MNEMÔNICO Destino, Operando1, Operando2      ;Comentário
```

- **Label** é um nome para referência de endereço pelo compilador
- **Mnemônico** é a instrução em si.
- **Operandos** são os registradores, dados , endereços manipulados, etc ...

Ex: `ADD R2, R4, R5` ;R2 = R4 + R5

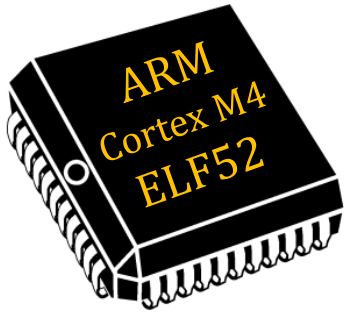
UTFPR

Cortex-M ISA

Endereçamento

- As instruções operam com **dados** e **endereços**
- Existem alguns **modos** ou formatos de representar estes **dados** e **endereços**. Isto reflete no formato que a instrução usa para identificar os valores e posições de memória, seja para ler ou escrever os dados.
- **Modos de Endereçamento**:

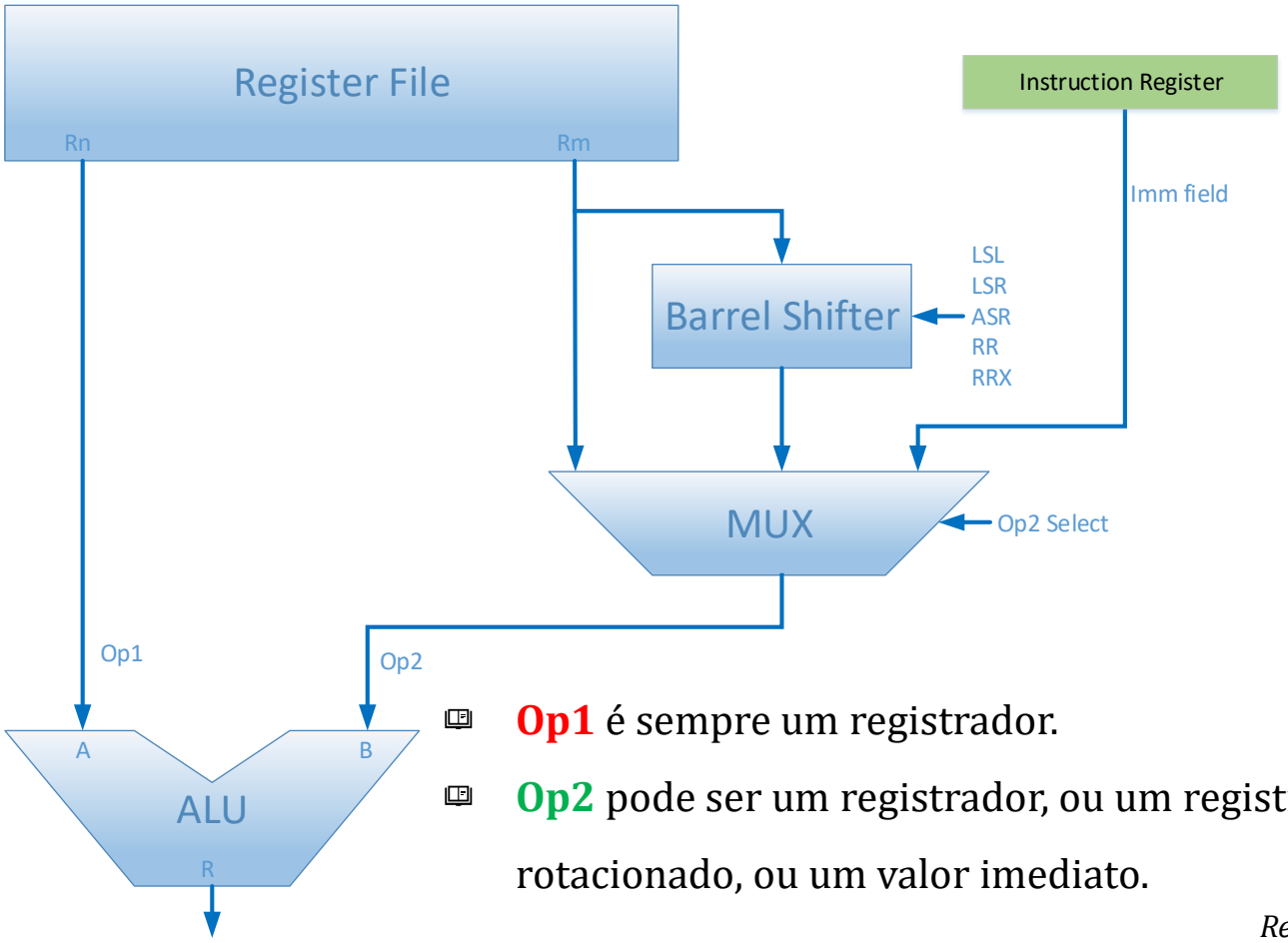
Imediato | Registrador | Indexado | Relativo



UTFPR

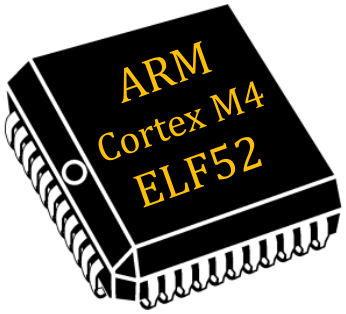
Cortex-M ISA

Origem dos operandos



- **Op1** é sempre um registrador.
- **Op2** pode ser um registrador, ou um registrador rotacionado, ou um valor imediato.

*Ref. **



Cortex-M ISA

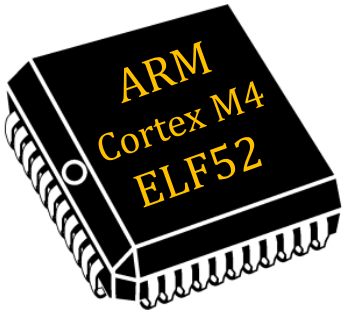
Origem dos operandos

Exemplos para **Op2**:

```
ADD   R2, R4, R5          ;Op2 é um registrador (R5)
ADD   R2, R4, R5,LSL #2   ;Op2 é um registrador rotacionado
                          ;R5 Logic Shifted Left 2-bit
                          ;equivalente a multiplicar por 4
ADD   R2, R4, #0xFF       ;Op2 é um valor imediato 0xFF
```

Base	Prefixo	Sufixo	Exemplo
Binário	2_	Y or y	2_1001 ou 1001Y
Decimal:	-	T or none	1234T or 1234
Hexadecimal:	0x or 0X	H or h	1234H or 0x1234
Octal:	(zero)	Q, q, O, or o	0777 or 777q or 777Q or 777o

https://developer.arm.com/documentation/101407/0537/Debugging/Expressions/Constants



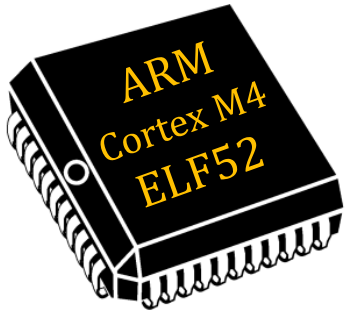
UTFPR

Cortex-M ISA

Possibilidades de deslocamentos

- ASR #n 1 ≤ n ≤ 32
- LSL #n 1 ≤ n ≤ 31
- LSR #n 1 ≤ n ≤ 32
- ROR #n 1 ≤ n ≤ 31
- RRx

Instrução	Descrição	#imm Sh
ASR Rd, Rn, Sh	Arithmetic Shift Right (preserves signal)	1..32
LSL Rd, Rn, Sh	Logical Shift Left	0..31
LSR Rd, Rn, Sh	Logical Shift Right	1..32
ROR Rd, Rn, Sh	Rotate Right	0..31
RRX Rd, Rn	Rotate Right Extended	
Sh	Pode ser valor imediato de 5-bit (1 a 32) ou (0 a 31)	
	Pode ser os 8-bit(0 a 255) menos significativos de um Registrador geral	



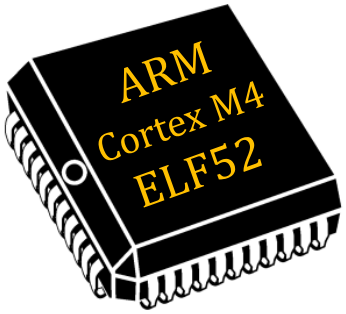
UTFPR

Cortex-M ISA

Operações de Transferência

- Para mover valores entre **registradores** ou uma **constante** e um **registrador**.
- Só conseguimos transferir no máximo valores de até **16 bits**.
- Exemplos:

```
MOV{S}{cond} Rd, <op2>      ;set Rd equal to the value specified
                            ;by op2
MOV{cond} Rd, #im16         ;set Rd equal to im16, im16 is 0 to
                            ;65535
MVN{S}{cond} Rd, <op2>      ;set Rd equal to the value specified
                            ;by op2
```



Cortex-M ISA

Operações de Transferência

- Quando desejar carregar um registrador com uma constante não suportada pelo **MOV**, o que fazer?
- USAR o **MOV** e **MOVT** (Instrução):
- Exemplos:

```
MOV  R1, #0x5678      ;R1[31:16]:=0, R1[15:0]:=0x5678
MOVT R1, #0x1234      ;R1[31:16]:=0x1234, R1[15:0]=mantém
```

- No Keil (Diretiva – *... não faz parte do assembly ...*):

```
LDR R6, Pi                ;Definição da constante fora da
                          ;execução do código
Pi  DCD 314159            ;Define Code ...
ou
LDR R6, =314159           ;Não é LOAD. Diretiva pré-compilação
```

UTFPR

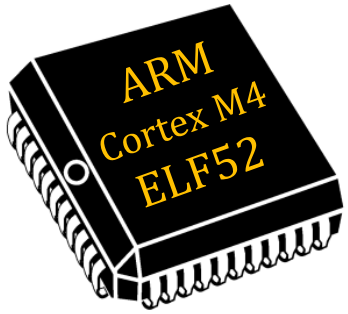
Cortex-M ISA

Constantes no Op2

- Podem variar entre **3** a **16-bit**
- Constantes de 8-bit são mais comuns podem:
  - Ser deslocadas ,,. (0 a 31-bit)
  - Padrões: **0xXYXYXYXY**, **0x00XY00XY** ou **0xXY00XY00**

- Exemplos:

```
MOVT    R0,#0xACFC          ;16-bit cte
MOV     R0,#0xE100E100      ;8-bit padrão
MOV     R0, R1, LSL #27     ;5-bit cte
```

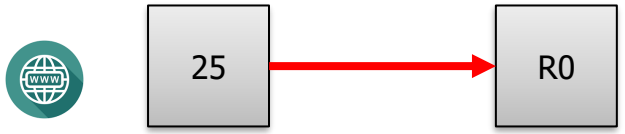


Cortex-M ISA

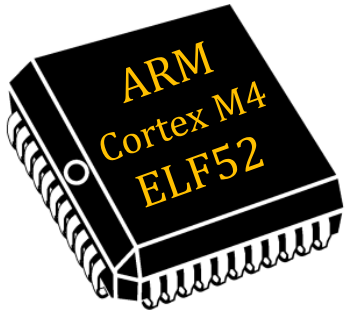
Endereçamento Imediato

- Uma constante pode ser colocada dentro do código de instrução
- Definido por uma "*hashtag* "('#') antes do operando.
- Exemplos:

```
MOV     R0, #25
```




```
MOV     R0, #25       ;move a cte dec. 25 para o reg R0
MOV     R1, #0x2F     ;move a cte hexa 2Fh para o reg R1
MOV     R2, #2_1101   ;move a cte binária 00001101 para R2
```

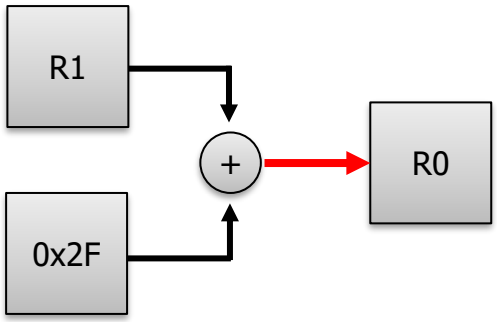


Cortex-M ISA

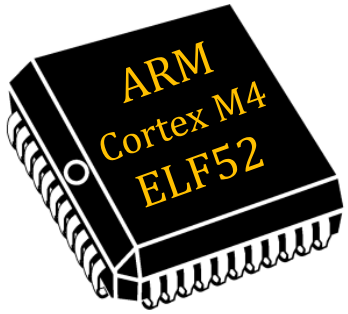
Endereçamento por Registrador

- Algumas instruções podem operar dados com registradores do microprocessador.
- Registrador com valor imediato
- Exemplos:

```
ADD R0, R1, #0x2F
```




```
ADD   R1, R2, #18        ;R1 <= R2 + 18
AND   R0, R1, #0x0F      ;R0 <= R1 & 0x0F
MUL   R0, R2, #8         ;R0 <= R2 * 8
```



UTFPR

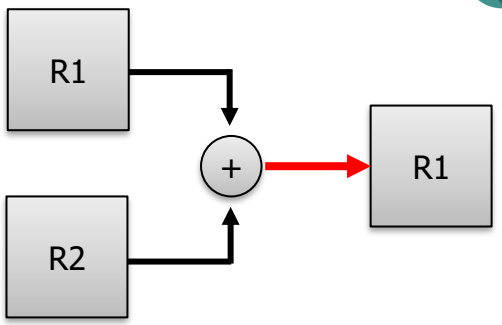
Cortex-M ISA

Endereçamento por Registrador

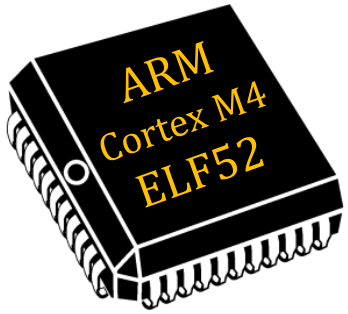
- Registrador com Registrador
- Exemplos:

```
MOV   R1, R2
```


```
ADD   R1, R2
```




```
ADD   R0, R1, R2      ;R0 <= R1 + R2
ADD   R3, R4          ;R3 <= R3 + R4
MOV   R1, R3          ;R1 <= R3
```

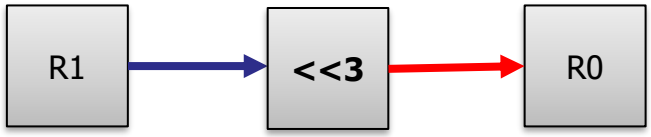


Cortex-M ISA

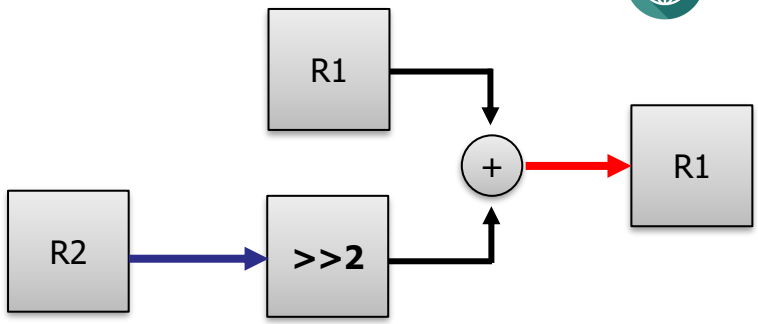
Endereçamento por Registrador

- Registrador com **Op2** deslocado
- Exemplos:

```
MOV R0, R1, LSL #3
```




```
ADD R0, R1, R2, LSR #2
```



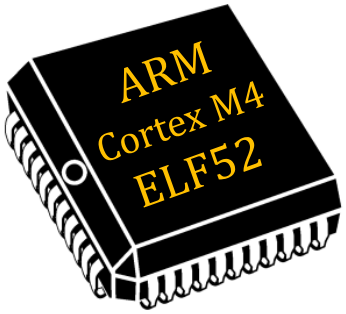

```
MOV   R0, R1, LSL #3      ;R0 <= (R1 << 3)
ADD   R0, R1, R2 LSR #2   ;R0 <= R1 + (R2 >> 2)
MOV   R1, R3, ASR #7      ;R1 <= R3 / 128 (signed)
MOV   R2, R4, LSR #7      ;R2 <= R4 / 128 (unsigned)
```

Cortex-M ISA

Endereçamento Indexado

- Instruções **ARM** não suportam operações de memória para memória (**RAM** ou **ROM**);
- Somente as instruções **LDR**/**STR** podem acessar a memória;
- Os registradores atuam como ponteiros para a memória;
- Exemplos:

```
LDR   R0, [R1]      ;R0<=[R1] R0 recebe o dado apontado por R1
LDR   R0, [R1, #0]  ;R0<=[R1+0] o mesmo que R0=[R1]
STR   R2, [R0, #4]  ;[R0+4]<=R2 guarda o dado R2 no endereço
                    ;apontado por [R0+4]
```



Cortex-M ISA

Endereçamento Relativo ao (PC)

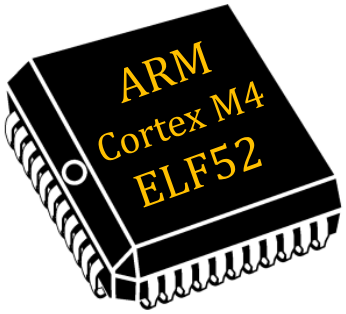
- Se o ponteiro for o *Program Counter* (**PC**)
- Usado para:
  - Saltos (***Branches***);
  - Chamadas de funções;
  - Acesso à constantes salvas na **ROM**
- Exemplos:

```
B     label          ;pula para label
BL    subrotina      ;chama subrotina e salva PC no R14 (LR)
BX    R14            ;return ou  MOV PC, R14
LDR   R1, =Count          ;R1 aponta para Count *(Diretiva)*
LDR   R0, [R1]       ;R0 <= valor apontado por R1
```

Cortex-M ISA

Operandos

Operando	Representa
Ra Rd Rm Rn Rt e Rt2	Registradores
{Rd,}	Registrador de destino opcional
#imm12	Constante de 12 bits, 0 a 4095
#imm16	Constante de 16 bits, 0 a 65535
operand2	Segundo operando flexível *
{cond}	Condição lógica opcional *
{type}	Estabelece um tipo de dado opcional *
{S}	Opcional que seta os bits de condição
Rm {, shift}	Deslocamento opcional no Rm
Rn {, offset}	Offset opcional no Rn



Cortex-M ISA

Operandos

Operando 2	
#constant	Valor imediato de 8 bits*
Rm {, <opsh>}	Registrador, deslocado opcionalmente como abaixo
Rm, LSL Rs	Registrador Rm com deslocamento lógico para esquerda definido por Rs
Rm, LSR Rs	Registrador Rm com deslocamento lógico para direita definido por Rs
Rm, ASR Rs	Registrador Rm com deslocamento aritmético para direita definido por Rs
Rm, ROR Rs	Registrador Rm com rotação lógica para direita definido por Rs

- Podem variar entre **3** a **16-bit**
- Constantes de **8-bit** são mais comuns podem:
  - Ser deslocadas ,,. (0 a 31-bit)
  - Padrões: **0xXYXYXYXY**, **0x00XY00XY** ou **0xXY00XY00**
- Exemplos:

```
MOVT    R0,#0xACFC          ;16-bit cte
MOV     R0,#0x00E100E1      ;8-bit padrão
MOV     R0,R1,LSL #27       ;5-bit cte
```

UTFPR

Cortex-M ISA

Acesso a Memória

- Acesso à memória de código ou de dados.
  - **LDR** lê dados da memória;
  - **STR** escreve dados na memória;
  - Instruções de processamento de dados não acessam a memória.
  - Arquitetura **LOAD-STORE (Risc)**

- Para operações com dados em memória:
  - Leitura da memória em registrador;
  - Operação;
  - Escrita em memória.

Cortex-M ISA

Acesso a Memória

- Para acessar a memória **SEMPRE** estabelecer um **registrador ponteiro (ou base)** para o objeto

- Exemplos (Estado Inicial):

Registradores

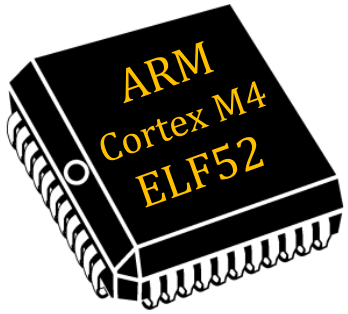
R0	0x20000000
R1	0x00
R2	0x00
R3	0x50003210
R4	0x00

...

Memória

0x20000000	0x44332211
0x20000004	0x88776655
0x20000008	0x12345678
0x2000000C	0xDDCCBBAA
0x20000010	0x99999999

...

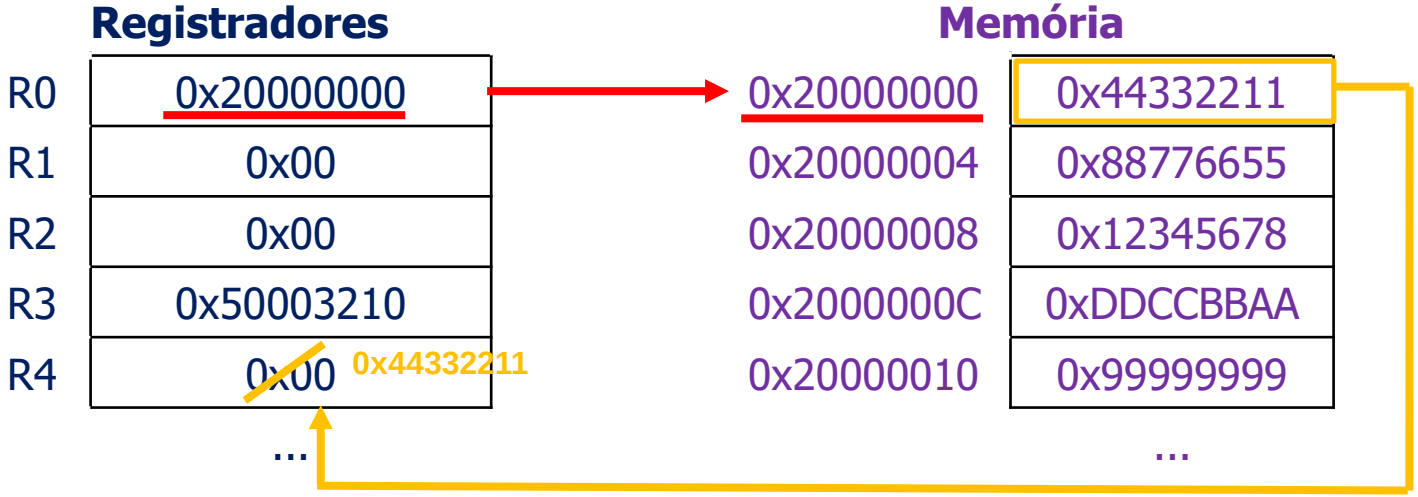


Cortex-M ISA

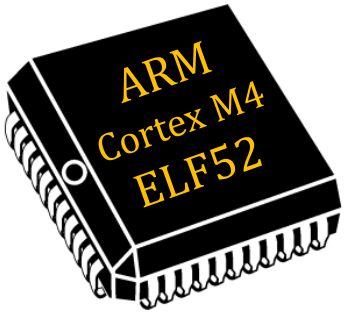
Acesso a Memória

Exemplo:

```
LDR    R4,  [R0]
```



Obs: O primeiro operando é por registrador e o segundo é indexado ([]).

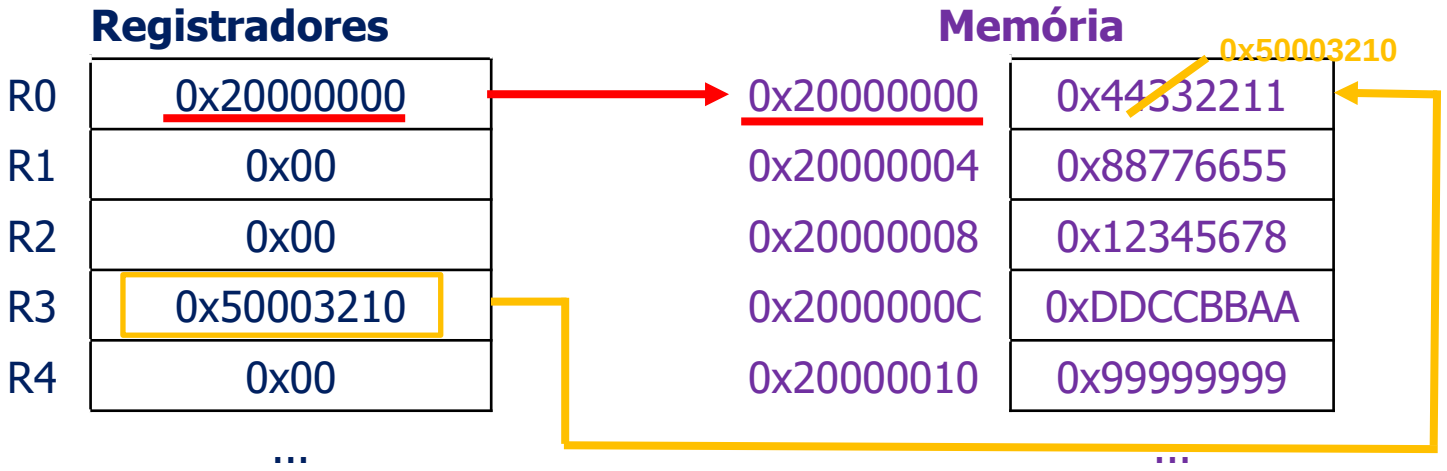


Cortex-M ISA

Acesso a Memória

Exemplo:

```
STR    R3, [R0]
```



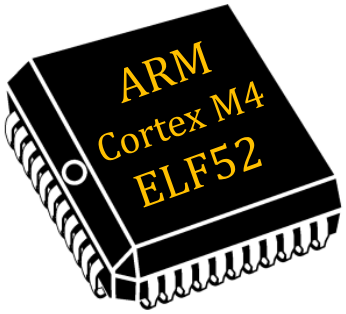
Obs: O primeiro operando é por registrador e o segundo é indexado ([]).

UTFPR

Cortex-M ISA

Tipos de Acesso a Memória

- O registrador que contém o endereço ou a localização dos dados é chamado de **registrador base**.
- Utiliza-se o modo de **endereçamento indexado**.
- Há três tipos: **Com offset**, **pré-indexado**, **pós-indexado**.



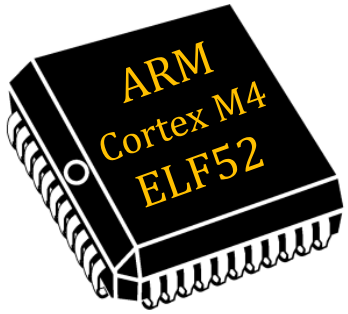
Cortex-M ISA

Tipos de Acesso a Memória

- Com ***offset***:
  - O endereço é incrementado **ANTES** da operação, mas o valor do registrador base **NÃO** é alterado.
  - [XX] ⇨ Conteúdo apontado por XX

- Exemplos:

```
LDR   R0, [R1]       ;R0<=[R1] R0 recebe o dado apontado por R1
LDR   R0, [R1, #8]   ;R0<=[R1+8] R0 recebe o dado apontado pelo
                     ;R1 + 8
STR   R2, [R0, #4]   ;[R0+4]<=R2 guarda o dado R2 endereço
                     ;apontado por R0 + 4
```

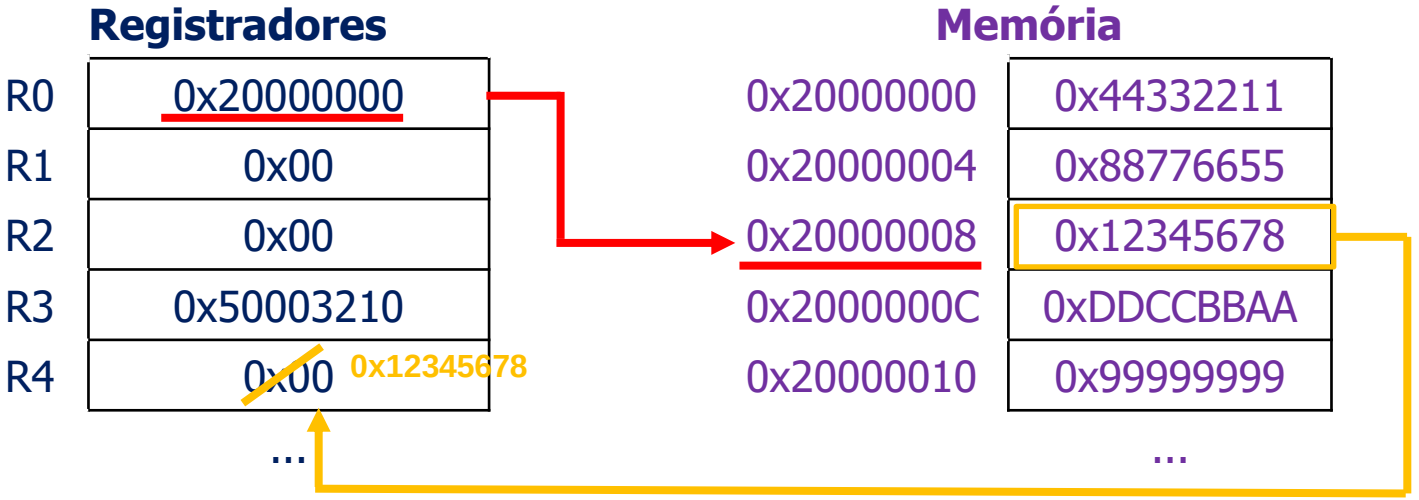


Cortex-M ISA

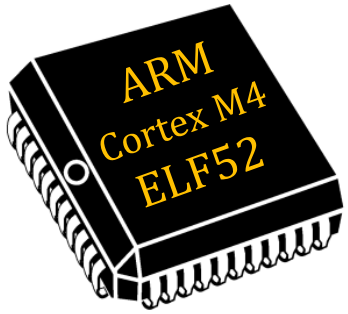
Acesso a Memória com *Offset*

Exemplo:

```
LDR    R4, [R0, #8]
```



Obs: O primeiro operando é por registrador e o segundo é indexado com offset ([]).



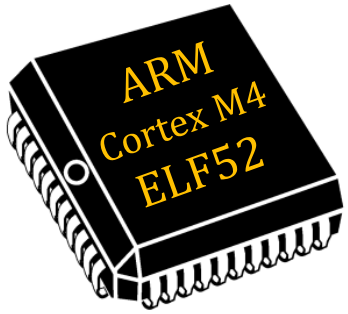
Cortex-M ISA

Tipos de Acesso a Memória

- Pré-Indexado ⇨ **'!'** (*writeback*)
  - O endereço é incrementado **ANTES** da operação, e o valor do **registrador base** é **atualizado**.

- Exemplos:

```
LDR   R0, [R1, #2]!        ;R0<=[R1+2], R1<=R1+2
STR   R2, [R3, #4]!        ;[R3+4]<=R2, R3=R3+4
LDR   R0, [R1, R2]!        ;R0<=[R1+R2], R1<=R1+R2
LDR   R0, [R1, R2, LSR #2]! ;R0<=[R1+(R2<<2)], R1<=R1+(R2<<2)
```

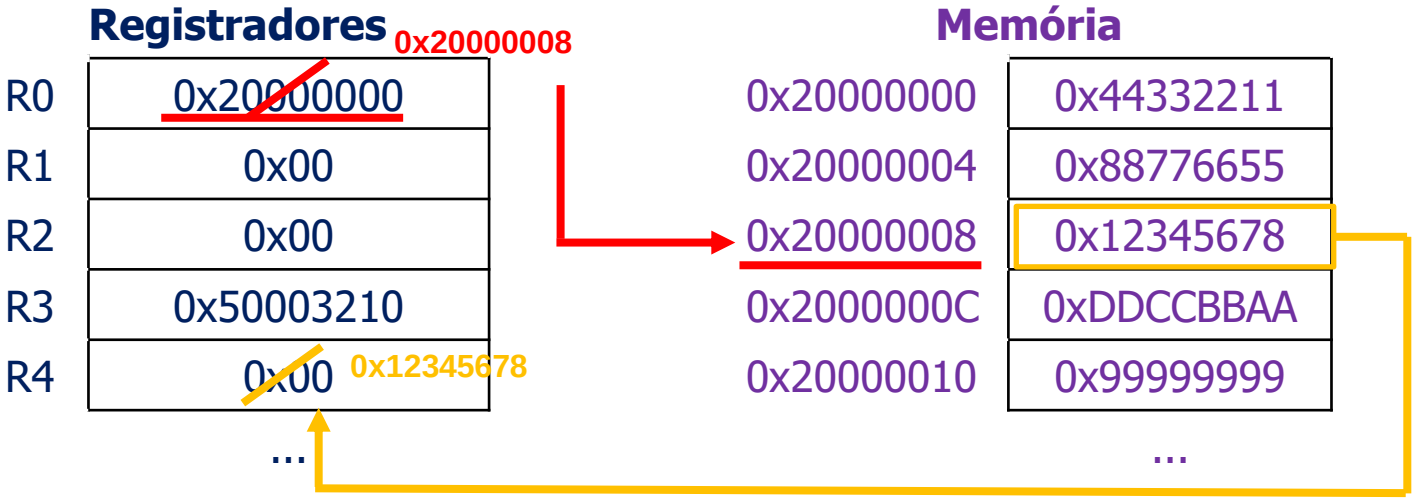


Cortex-M ISA

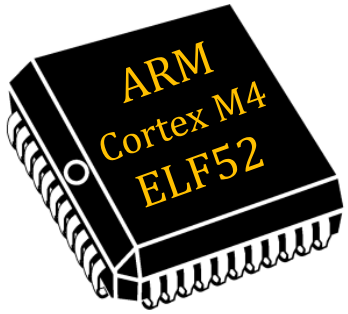
Pré - Indexado

Exemplo:

```
LDR   R4, [R0, #8]!
```



*Obs: O primeiro operando é por registrador e o segundo é indexado com offset ([]) e writeback **!**.*



Cortex-M ISA

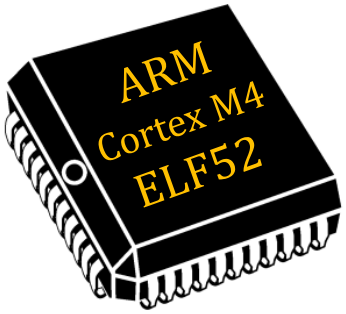
Tipos de Acesso a Memória

- Pós-Indexado:

  - O endereço é incrementado **APÓS** a operação, e o valor do **registrador base** é **atualizado**.

- Exemplos:

```
LDR   R0, [R1], #8          ;R0<=[R1], R1<=R1+8
STR   R2, [R3], R4          ;[R3]<=R2, R3=R3+R4
LDR   R0, [R1], R2, LSR #2  ;R0<=[R1], R1<=R1+(R2<<2)
```

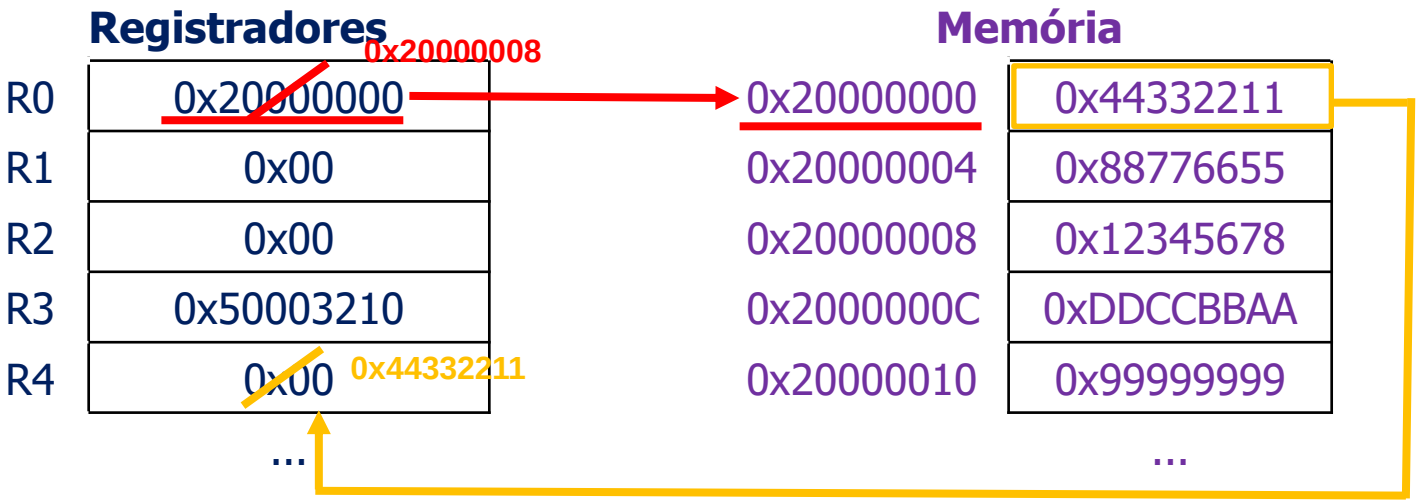


Cortex-M ISA

Pós - Indexado

Exemplo:

```
LDR   R4, [R0], #8
```

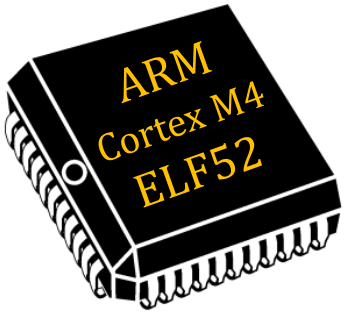


Cortex-M ISA

Acesso à Memória

Tabela comparativa:

Modo	Exemplo	Resultado	Alteração Op2
Pré com writeback (!)	LDR R0,[R1,#4] !	R0 = [R1 + 4]	R1 = R1 + 4
	LDR R0,[R1,R2] !	R0 = [R1+R2]	R1 = R1 + R2
	LDR R0,[R1,R2,LSL #2] !	R0 = [R1 + (R2 << 2)]	R1 = R1 + R2 << 2
Pré-indexado	LDR R0,[R1,#4]	R0 = [R1 + 4]	sem alteração
	LDR R0,[R1,R2]	R0 = [R1+R2]	sem alteração
	LDR R0,[R1,R2,LSL #2]	R0 = [R1 + (R2 << 2)]	sem alteração
Pós-indexado	LDR R0,[R1],#4	R0 = [R1]	R1 = R1 + 4
	LDR R0,[R1],R2	R0 = [R1]	R1 = R1 + R2
	LDR R0,[R1],R2,LSL #2	R0 = [R1]	R1 = R1 + R2 << 2



Cortex-M ISA

Algumas variações *Load/Store*

Exemplos:

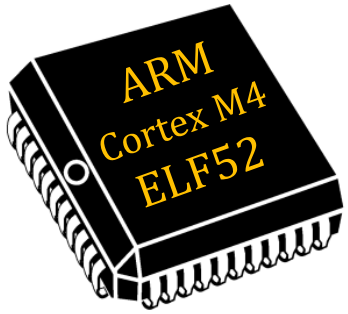
```
LDR{type}{cond} Rd, [Rn]        ;load memory at [Rn] to Rd
STR{type}{cond} Rt, [Rn]        ;store Rt to memory at [Rn]
LDR{type}{cond} Rd, [Rn, #n]    ;load memory at [Rn+n] to Rd
STR{type}{cond} Rt, [Rn, #n]    ;store Rt to memory [Rn+n]
LDR{type}{cond} Rd, [Rn, #n]!   ;load memory at [Rn+n] to Rd;
                                ;Rn<=Rn+n;
STR{type}{cond} Rt, [Rn, #n]!   ;store Rt to memory [Rn+n]
                                ;Rn<=Rn + n;
LDR{type}{cond} Rd, [Rn], #n    ;load memory at [Rn] to Rd;
                                ;Rn<=Rn + n;
STR{type}{cond} Rt, [Rn], #n    ;store Rt to memory [Rn]
                                ;Rn<=Rn + n;
```

Cortex-M ISA

Tipos de Dados

- Em relação a dados de memória, pode-se acessar dados de **8**, **16**, **32** ou **64** bits.
- Ao colocar um valor de **8** bits ou **16** bits em um registrador, os bits mais significantes são preenchidos com **0**.

{type}	Tipo do dado	Significado
	Word de 32 bits	0 a +4.294.967.295 ou -2.147.483.648 a +2.147.483.647
B	Byte de 8 bits sem sinal	0 a 255
SB	Byte de 8 bits com sinal	-128 a +127
H	Halfword de 16 bits sem sinal	0 a 65535
SH	Halfword de 16 bits com sinal	-32768 a +32767
D	64-bits	Usa dois registradores



UTFPR

Cortex-M ISA

Little / Big endian

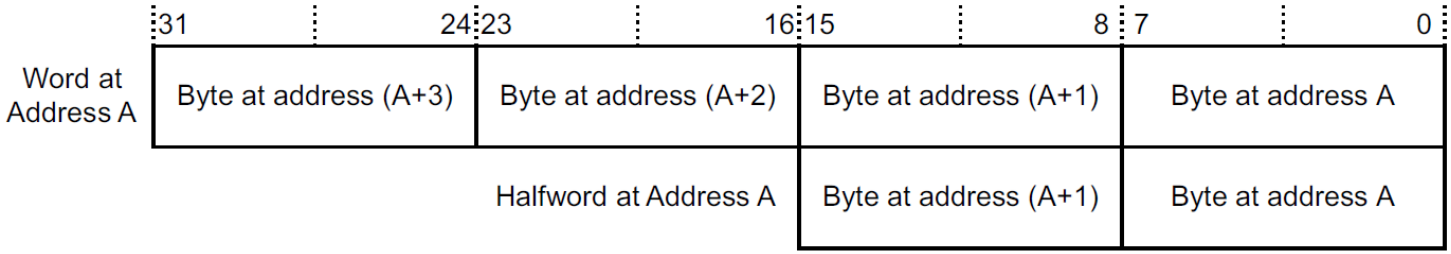


Figure A3-1 Little-endian byte format

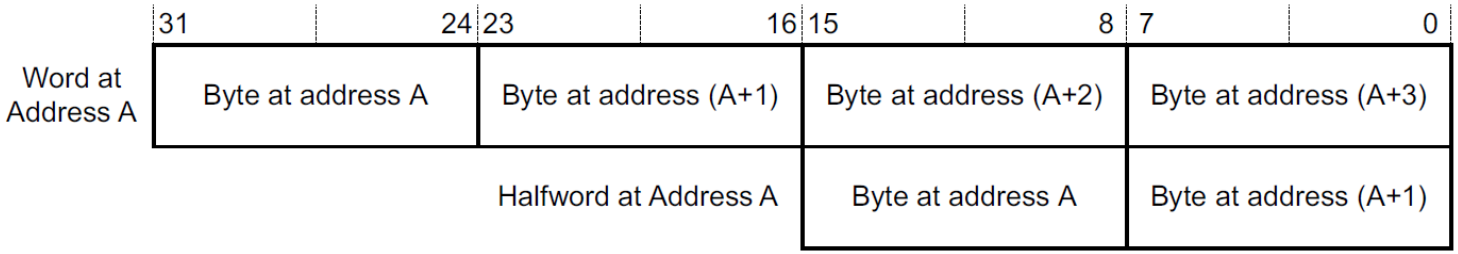
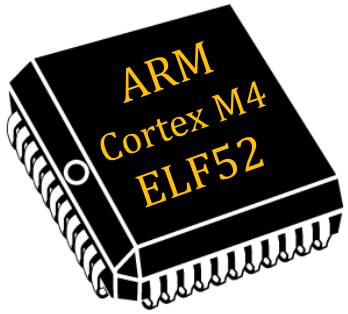


Figure A3-2 Big-endian byte format

*Ref. **



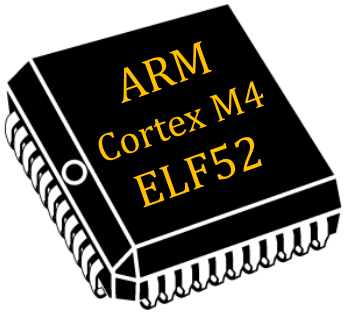
Cortex-M ISA

Little endian

MSByte	MSByte-1	LSByte+1	LSByte
Word at address A			
Halfword at address (A+2)		Halfword at address A	
Byte at address (A+3)	Byte at address (A+2)	Byte at address (A+1)	Byte at address A

Figure A3-3 Little-endian memory system

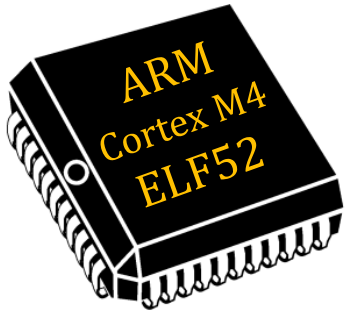
*Ref. **



Cortex-M ISA

Acesso a Memória

Instrução	Descrição
LDRB	Load byte. Read a byte from memory, zero extend and store in register.
LDRH	Load half word. Read a half-word from memory, zero-extend and store in register.
LDR	Load register. Read a word from memory and store in a register.
LDRD	Load double. Read a double word form memory and store in two registers.
STRB	Store byte. Store the LSB of a register into memory (byte wide)
STRH	Store half-word. Store the lower half of a register into memory (16-bit wide)
STR	Store register. Store a register into memory (32-bit wide)
STRD	Store double. Store the two registers into memory (64-bit wide)
LDM	Load multiple. Read several (up to 16) registers from memory.
STM	Store multiple. Store several (up to 16) registers into memory.
PUSH	Store registers in current stack (main or process)
POP	Restore registers in current stack (main or process)
LDREX	Load Register Exclusive
LDREXB	Load Byte to Register Exclusive
LDREXH	Load Half Word to Register Exclusive
STREX	Store Register Exclusive
STREXB	Store Register Exclusive to Byte
STREXH	Store Register Exclusive to Half Word
CLREX	Clear local processor exclusive tag
LDRT	Load unprivileged. With variants for Byte and HalfWord.
STRT	Store unprivileged. With variants for Byte and HalfWord.



Cortex-M ISA

Acesso a Memória

Exemplo:

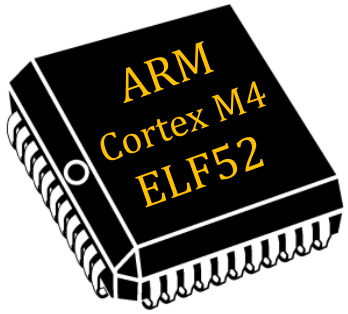
```
LDRB    R4,  [R0]
```

Registradores

R0	0x20000002
R1	0x00
R2	0x00
R3	0x50003210
R4	0x00 → 0x33
...	

Memória

0x20000000	0x11	0x22	0x33	0x44
0x20000004	0x55	0x66	0x77	0x88
0x20000008	0x78	0x56	0x34	0x12
0x2000000C	0xAA	0xBB	0xCC	0xDD
0x20000010	0x99	0x99	0x99	0x99
...				

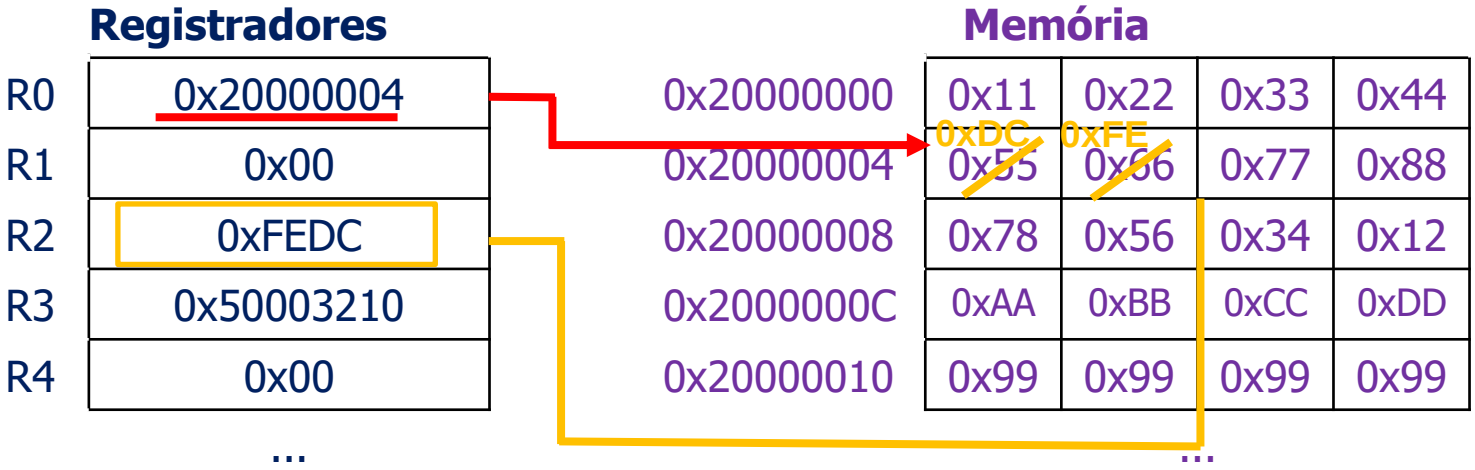


Cortex-M ISA

Acesso a Memória

Exemplo:

```
STRH    R2,  [R0]
```



Cortex-M ISA

Tutorial criar um projeto no Keil

Tutorial Prof. Peron (11pgs):
Adaptado por Prof. DaLuz
http://www.elf52.daeln.com.br/Pdfs/Tutorial_Novo_Projeto_Keil.pdf

UTFPR

Cortex-M ISA

Exercício

1. Faça um código que realize os seguintes passos e depois depure no Keil: *(Acrescentar ao final do arquivo a instrução **NOP** para conseguir depurar o código inteiro. Esta instrução significa **No Operation**.)*

a) Salvar no registrador **R0** o valor **65** decimal

b) Salvar no registrador **R1** o valor **0x1B00.1B00**

c) Salvar no registrador **R2** o valor **0x1234.5678**

d) Guardar na posição de memória **0x2000.0040** o valor de **R0**

e) Guardar na posição de memória **0x2000.0044** o valor de **R1**

f) Guardar na posição de memória **0x2000.0048** o valor de **R2**

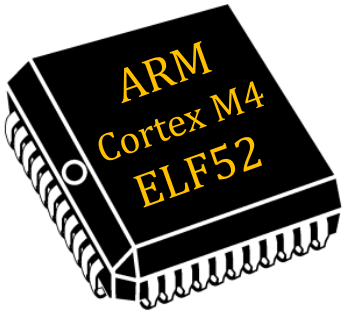
g) Guardar na posição de memória **0x2000.004C** o número **0xF0001**

h) Guardar na posição de memória **0x2000.0046** o **byte 0xCD**, sem sobrescrever os outros bytes da **WORD**

i) Ler o conteúdo da memória cuja posição **0x2000.0040** e guardar no **R7**

j) Ler o conteúdo da memória cuja posição **0x2000.0048** o guardar **R8**

k) Copiar para o **R9** o conteúdo de **R7**



Referências:

Atividade Prática 0:
Adaptado por Prof. DaLuz
http://www.elf52.daeln.com.br/Labs/Laboratorio_AP0.pdf

* *Refs* ↔ Renesas.com, Pixabay.com, wikimedia.org, flickr, community.arm.com, Undergraduated course Renesas (Prof. Douglas P. B. Renaux e Robson Linhares), ytchannel Gustavo W. Dernardin. *ARMv7-M Architecture Reference Manual*