

- Assembly

- Referências

Microcontroladores: (LT36D)

Prof: DaLuz



- Assembly

- Referências

Cortex-M ISA

Operações Lógicas

- 📖 Combinar, modificar, extrair ou testar bits de informações nos registradores.
- 📖 Operações **Unárias** (uma entrada):
 - **Negação**;
 - **Complementar**;
- 📖 Operações **Binárias** (duas entrada):
 - **AND**;
 - **OR**;
 - **XOR**;



- Assembly

- Referências

Cortex-M ISA

Operações Lógicas

- Algunas instruções lógicas
- Podemos adicionar o sufixo “S” para as condições N ou Z serem atualizadas durante a operação
- Exemplos:

AND{S}{cond} {Rd,} Rn, <op2>



; Rd=Rn&op2

ORR{S}{cond} {Rd,} Rn, <op2>

; Rd=Rn|op2



EOR{S}{cond} {Rd,} Rn, <op2>



; Rd=Rn^op2

BIC{S}{cond} {Rd,} Rn, <op2>

; Rd=Rn& (~op2)



ORN{S}{cond} {Rd,} Rn, <op2>



; Rd=Rn| (~op2)



Cortex-M ISA

Exercício

2. 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**.)*

- Realizar a operação lógica **AND** do valor **0xF0** com o valor binário **01010101** e salvar o resultado em **R0**. Utilizar o sufixo '**S**' para atualizar os *flags*;
- Realizar a operação lógica **AND** do valor **11001100** binário com o valor binário **00110011** e salvar o resultado em **R1**. Utilizar o sufixo '**S**' para atualizar os *flags*;
- Realizar a operação lógica **OR** do valor **10000000** binário com o valor binário **00110111** e salvar o resultado em **R2**. Utilizar o sufixo '**S**' para atualizar os *flags*;
- Realizar a operação lógica **AND** do valor **0xABCDABCD** com o valor **0xFFFF0000** (sem usar **LDR-diretiva**) e salvar o resultado em **R3**. Utilizar o sufixo '**S**' para atualizar os *flags*. Utilizar a instrução **BIC**;



- Assembly

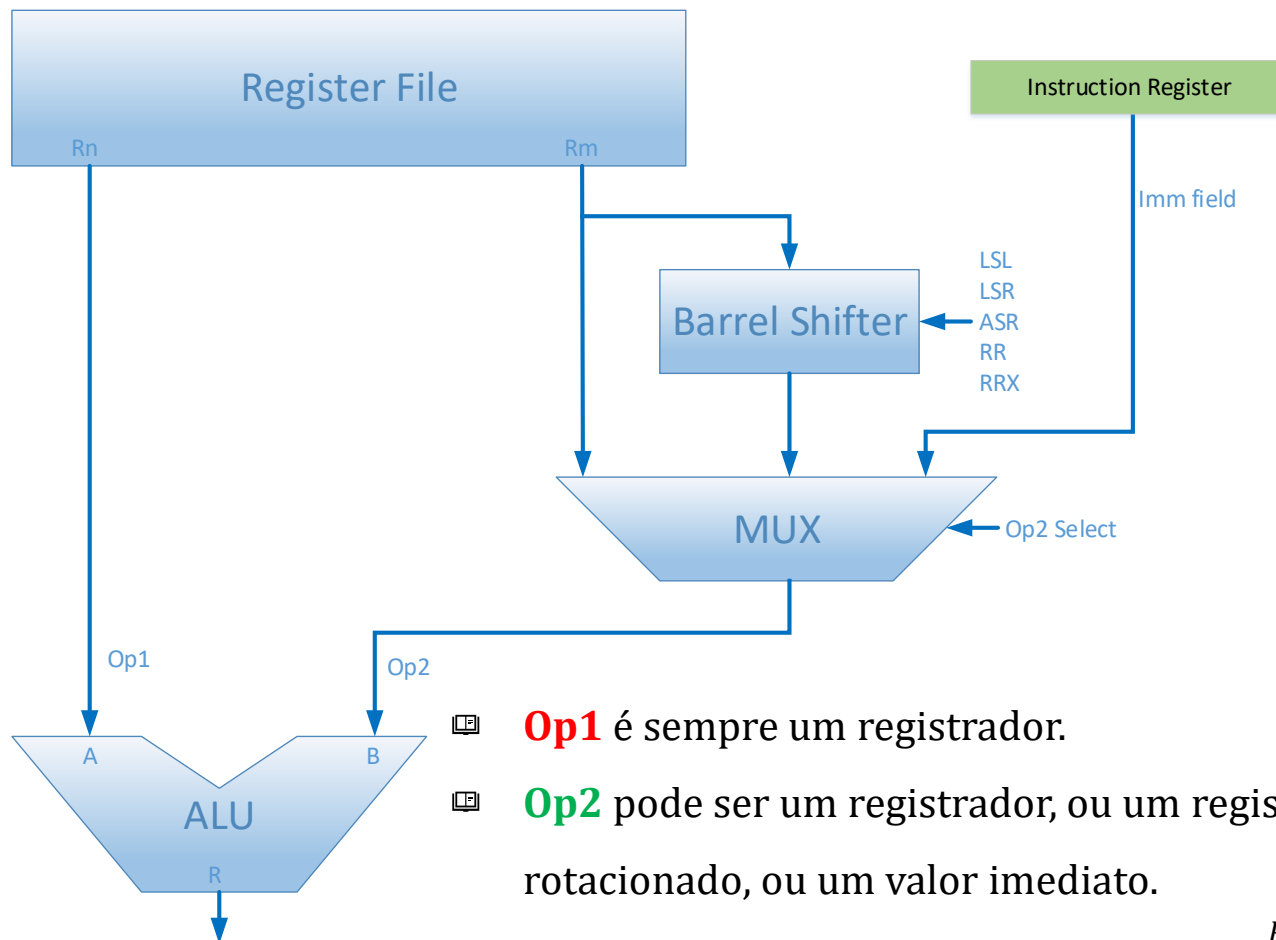
- Referências



- Assembly
- Referências

Cortex-M ISA

Origem dos operandos



Ref. *



Cortex-M ISA

Origem dos operandos



- Assembly

- Referências

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

Obs: #valor \Rightarrow #-15, #0xCFC, #2_1010 ...





Cortex-M ISA

Possibilidades de deslocamentos

 ASR #n

$1 \leq n \leq 32$



 LSL #n

$1 \leq n \leq 31$



 LSR #n

$1 \leq n \leq 32$



 ROR #n

$1 \leq n \leq 31$



 RRX



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>



Cortex-M ISA

Instruções de Deslocamento no Op2

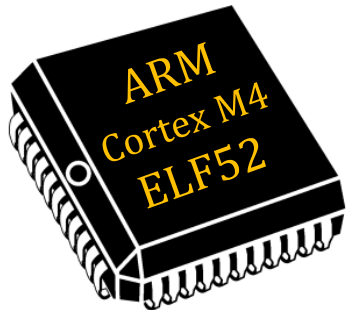


- Assembly

- Referências

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	



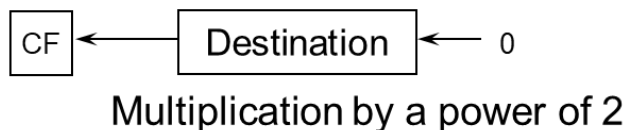


- Assembly
- Referências

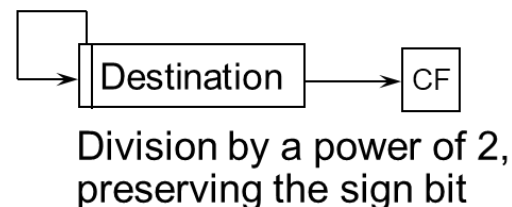
Cortex-M ISA

Instruções de Deslocamento

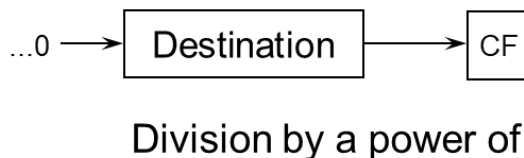
LSL : Logical Left Shift



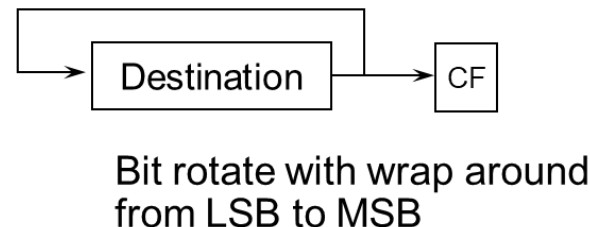
ASR: Arithmetic Right Shift



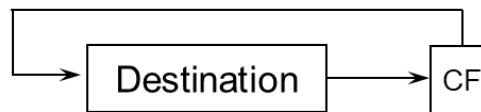
LSR : Logical Shift Right



ROR: Rotate Right



RRX: Rotate Right Extended



Ref. *



Cortex-M ISA

Exercício

3. 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. Verifique na simulação os valores dos registradores antes e depois)*

- Realizar o deslocamento **lógico** em 5 bits do número **701** para a direita com o *flag 'S'*;
- Realizar o deslocamento lógico em 4 bits do número **-32067** para a direita com o *flag 'S'*; (Usar o **MOV** para o número positivo e depois **NEG** para negativar)
- Realizar o deslocamento **aritmético** em 3 bits do número **701** para a direita com o *flag 'S'*;
- Realizar o deslocamento **aritmético** em 5 bits do número **-32067** para a direita com o *flag 'S'*;
- Realizar o deslocamento **lógico** em 8 bits do número **255** para a esquerda com o *flag 'S'*;
- Realizar o deslocamento **lógico** em 18 bits do número **-58982** para a esquerda com o *flag 'S'*;
- Rotacionar em 10 bits o número **0xFABC.1234**;
- Rotacionar em 2 bits com o *carry* o número **0x0000.4321**; (Realizar duas vezes)



- Assembly

- Referências



Cortex-M ISA

Aritméticas



- Assembly

- Referências

Instrução	Descrição	Operação
ADD Rd, Rn, Op2	Add a register to Op2	$Rd = Rn + Op2$
ADC Rd, Rn, Op2	Add a register to Op2 and to Carry	$Rd = Rn + Op2 + CY$
SUB Rd, Rn, Op2	Subtract from a register the Op2	$Rd = Rn - Op2$
SBC Rd, Rn, Op2	Subtract from a register the Op2 and the Borrow (negation of Carry)	$Rd = Rn - Op2 - /CY$
RSB Rd, Rn, Op2	Subtract from Op2 a register	$Rd = Op2 - Rn$
RSC Rd, Rn, Op2	Subtract from Op2 a register and the Borrow	$Rd = Op2 - Rn - /CY$
MOV Rd, Op2	Move to Rd from Op2 (put a copy of Operand2 into Rd)	$Rd = Op2$
MVN Rd, Op2	Move to Rd /Op2	$Rd = /Op2$
MOVT Rd,<imm16>	Move to Rd[31:16] from imm16. Lower bits of Rd are unaffected	$Rd[31:16] = imm16$



- **Carry:**

- Soma:
 - 0: soma coube nos 32 bits
 - 1: soma não coube nos 32bits
- Subtração:
 - 1: resultado positivo ou zero
 - 0: resultado negativo

- **Overflow:**

- Operações com sinal
- O bit V é setado quando há uma passagem entre 0x8000.0000 e 0x7FFF.FFFF



Cortex-M ISA

Comparação e teste



- Assembly

- Referências

Instrução	Descrição
CMP Rn, Op2	Compare: Subtract from Rn the Op, discard result, change flags
CMN Rn, Op2	Compare negative: Add Rn to Op2, discard result, change flags
TST Rn, Op2	Test: Rn AND Op2, discard result, change flags
TEQ Rn, Op2	Test equivalence: Rn EOR Op2, discard result, change flags





Cortex-M ISA

Multiplicação e Divisão



- Assembly

- Referências

Instrução	Descrição	Operação
<i>Instructions that multiply 32-bit by 32-bit resulting 32-bit with wrapping (LSW is preserved and higher bits are discarded)</i>		
MUL Rd, Rm, Rs	Multiply	$Rd = Rm * Rs$
MLA Rd, Rm, Rs, Rn	Multiply and accumulate	$Rd = Rm * Rs + Rn$
MLS Rd, Rm, Rs, Rn	Multiply and subtract	$Rd = Rm * Rs - Rn$
<i>Long multiplication: multiply 32-bit by 32-bit resulting 64-bit</i>		
UMULL RdLo, RdHi, Rm, Rs	Unsigned long multiply	$RdHi:RdLo = \text{unsigned}(Rm * Rs)$
UMLAL RdLo, RdHi, Rm, Rs	Unsigned long multiply and accumulate	$RdHi:RdLo = \text{unsigned}(RdHi:RdLo + Rm * Rs)$
UMAAL RdLo, RdHi, Rm, Rs	Unsigned long multiply and accumulate double	$RdHi:RdLo = \text{unsigned}(RdHi + RdLo + Rm * Rs)$
SMULL RdLo, RdHi, Rm, Rs	Signed long multiply	$RdHi:RdLo = \text{signed}(Rm * Rs)$
SMLAL RdLo, RdHi, Rm, Rs	Signed long multiply and accumulate	$RdHi:RdLo = \text{signed}(RdHi:RdLo + Rm * Rs)$



Instrução	Descrição	Operação
UDIV Rd, Rn, Rm	Unsigned divide	$Rd = Rn / Rm$
SDIV Rd, Rn, Rm	Signed divide	$Rd = Rn / Rm$





Cortex-M ISA

Exercício

4. 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). Verifique na simulação os valores dos registradores antes e depois)*

- a) Adicionar os números **101** e **253** atualizando os *flags*;
- b) Adicionar os números **1500** e **40543** sem atualizar os *flags*;
- c) Subtrair o número **340** pelo número **123** atualizando os *flags*;
- d) Subtrair o número **1000** pelo número **2000** atualizando os *flags*;
- e) Multiplicar o número **54378** por **4**; (Essa operação é semelhante a qual?)
- f) Multiplicar com o resultado em **64 bits** os números **0x1122.3344** e **0x4433.2211**;
- g) Dividir o número **0xFFFF.7560** por **1000** com sinal;
- h) Dividir o número **0xFFFF.7560** por **1000** sem sinal;



- Assembly

- Referências



Cortex-M ISA

Conjunto de bits



- Assembly

- Referências

Instrução	Descrição	Operação
BFC Rd,#<lsb>,#<width>	Bit field clear	clear Rd[(width+lsb-1)..lsb], others unchanged
BFI Rd, Rn,#<lsb>,#<width>	Bit field insert. Copy the <width> LSb of Rn to Rd	$Rd[(width+lsb-1)..lsb] = Rn[(width-1)..0]$
SBFX Rd, Rn,#<lsb>,#<width>	Signed bit field extract.	Copy bitfield from Rn to LSb of Rd and sign extend.
UBFX Rd, Rn,#<lsb>,#<width>	Unsigned bit field extract.	Copy bitfield from Rn to LSb of Rd and zero extend.



Um “*bit field*” é um conjunto de bits dentro de um registrador.

“*width*”=tamanho, n. de bits do conjunto (1 a 32)

“*lsb*”=posição do bit menos significativo do conjunto.



Cortex-M ISA

Extensão S/U



- Assembly

- Referências

Instrução	Descrição	Operação
SXTB Rd, Rm	Sign extend a byte	extract bits [7:0] and sign extend to 32 bits
UXTB Rd, Rm	Zero extend a byte	extract bits [7:0] and zero extend to 32 bits
SXTH Rd, Rm	Sign extend a half word	extract bits [15:0] and sign extend to 32 bits
UXTH Rd, Rm	Zero extend a half word	extract bits [15:0] and zero extend to 32 bits





Cortex-M ISA

Miscellaneous



- Assembly

- Referências

Instrução	Descrição
CPSID	Change Processor State, Disable Interrupts.
CPSIE	Change Processor State, Enable Interrupts.
DMB	Data Memory Barrier.
DSB	Data Synchronization Barrier.
ISB	Instruction Synchronization Barrier.
MRS	Move to Register from Special Register.
MSR	Move to Special Register from Register.
NOP	No Operation.
SVC	Supervisor Call.
WFI	Wait for Interrupt.





Cortex-M ISA

Miscellaneous

- ❏ **NOP:** *No operation, pode fazer o micro esperar 1 clock.* Pode ser removida do *pipeline*. Outro uso: alinhamento de código.

NOP



; 1 ciclo de máquina em espera

- ❏ **SVC:** *Supervisor call (S.O.)*

SCV #imm



; Gera uma chama de rotina de exceção.

- ❏ **WFI:** Usada para colocar o micro em *sleep mode*.

WFI



; Sleep até IRQ ou Exceção ou Debug

UTFPR

- Assembly

- Referências



Cortex-M ISA

Bloco IT

- ▣ Pode executar uma sequência de **1** a **4** instruções
- ▣ **ITxyz COND**
- ▣ onde **x, y, z** são **T** ou **E** (Then ou Else) e correspondem a uma instrução dentro do bloco.

Exemplos:

IT**TE** **EQ**  ;Bloco IT com 3 instruções

ADD**EQ** R2, R3, R5 ;Se Z=1, executa

SUB**EQ** R7, R8 ;Se Z=1, executa

ADD**NE** R2, R3, R5 ;Se Z=1, não executa



Cortex-M ISA

Bloco IT

Observações:

- Um salto **BCC** condicional pode estar dentro de um bloco **IT**.
- As instruções: **IT**, **CBZ**, **CBNZ**, **CPSIE**, **CPSID** não podem estar presentes em um bloco **IT**.
- Instruções que alteram o **PC** só podem aparecer por último num bloco **IT**.



- Assembly

- Referências



Cortex-M ISA

Condicionais - sufixos



- Assembly

- Referências

Sufixo	Flags	Significado	
EQ	Z = 1	Equal - Igual	S/U
NE	Z = 0	Not equal - diferente	S/U
CS or HS	C = 1	Higher or same - maior ou igual	U
CC or LO	C = 0	Lower - menor	U
MI	N = 1	Negative - negativo	S
PL	N = 0	Positive or zero - positivo ou zero	S
VS	V = 1	Overflow - estouro de bit	S
VC	V = 0	No overflow - sem estouro	S
HI	C = 1 e Z = 0	Higher - mior	U
LS	C = 0 ou Z = 1	Lower or same - menor ou igual	U
GE	N = V	Greater than or equal - maior que ou igual	S
LT	N != V	Less than - menor que	S
GT	Z = 0 e N = V	Greater than - maior que	S
LE	Z = 1 ou N != V	Less than or equal - menor que ou igual	S
AL	<i>any value</i>	Always - Sempre	S/U



Cortex-M ISA

Exercício

5. 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). Verifique na simulação os valores dos registradores antes e depois)*

- a) Mova o valor **10** para o registrador **R0**;
- b) Teste se o registrador é maior ou igual que **9**;
- c) Crie um bloco com **If-Then** com **3** execuções condicionais
 - Se sim, salve o número **50** no **R1**
 - Se sim, adicione **32** com o **R1** e salve o resultado em **R2**
 - Se não, salve o número **75** no **R3**
- d) Agora verifique se o registrador é maior ou igual a **11** e execute novamente o passo **(c)**;



- Assembly

- Referências



Cortex-M ISA

PUSH e POP

- Transferência de registradores para a pilha *Last-In-First-Out(LIFO)*:

PUSH {R0-R7}

POP {R1, R3-R6}

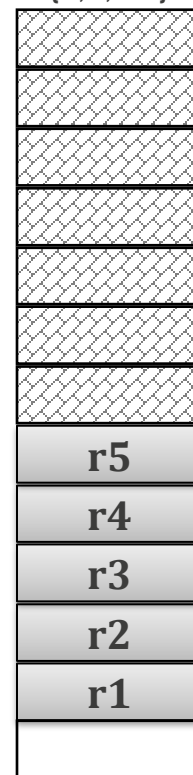


- Obs:** Caso o **PC** esteja na lista, um salto será executado, quando terminar o **POP**.

O **T-bit** do **xPSR** vem do **b0** do **PC**, deve ser **1** Cortex-M para que não gere **exceção**.

Endereço: **H**.

STMFD sp!, {r0,r1,r3-r5}
PUSH {r0,r1,r3-r5}



Antes: **SP** →

Depois: **SP** →

FD : Full Descending

Ref. *

UTFPR

- Assembly

- Referências



Cortex-M ISA

Exercício

6. Faça um código que realize os seguintes passos e depois depure no Keil: *(Acrescentar ao final do arquivo a instrução **NOP**)*
- a) Mover o valor **10** para o registrador **R0**;
 - b) Mover o valor **0xFF11.CC22** para o registrador **R1**;
 - c) Mover o valor **1234** para o registrador **R2**;
 - d) Mover o valor **0x300** para o registrador **R3**;
 - e) Empurrar para a pilha o **R0**;
 - f) Empurrar para a pilha os **R1, R2 e R3**;
 - g) Visualizar a pilha na memória (o topo da pilha está em **0x2000.0400**);
 - h) Mover o valor **60** para o registrador **R1**;
 - i) Mover o valor **0x1234** para o registrador **R2**;
 - j) Desempilhar corretamente os valores para os registradores **R0, R1, R2 e R3**;



- Assembly

- Referências



Cortex-M ISA

Saltos - Branches



- Assembly

- Referências

Instrução	Descrição	Distância
B <label>	Branch to target address.	+/-16 MB
Bcc <label>	Conditional Branch outside IT block	+/-1 MB
	Conditional Branch inside IT block	+/-16 MB
CBNZ Rn,<label>	Compare and Branch on Nonzero.	0-126 B
CBZ Rn,<label>	Compare and Branch on Zero.	
BL <label>	Call a subroutine.	+/-16 MB
BLX <register>	Call a subroutine, optionally change instruction set.	Any
BX <register>	Branch to target address, optionally change instruction set.	Any
TBB [Rn,Rm,LSL #1]	TBB: Table Branch, byte offsets. Base addr, index, optional #1	0-510 B
TBH [Rn,Rm,LSL #1]	TBH: Table Branch, halfword offsets.	0-131070 B

📖 **Obs:** O T-bit do xPSR vem do b0 do PC, deve ser 1 Cortex-M para que não gere **exceção**.



📖 **MOV PC,... - ADD PC, ... - POP {PC}** – quando executadas também executam saltos!

📖 Levam de 1 a 4 clocks e causam “flush” no pipeline.





- Assembly

- Referências

Cortex-M ISA

Saltos - Branches

Start

MOV R0, #4	rom: 0x08	PC=0x0C
ADD R6, R0, #10	rom: 0x0C	PC=0x10
LSL R2, R6, #2	rom: 0x10	PC=0x14
B pula1	rom: 0x14	PC=0x20
SUB R0, R1, #3	rom: 0x18	
ADD R0, R2, R5	rom: 0x1C	

pula1

MOV R3, #3	rom: 0x20	PC=0x24
MOV R2, #2	rom: 0x24	PC=0x28



- Assembly

- Referências

Cortex-M ISA

Saltos - Branches

Start

MOV R0, #4	rom: 0x08	PC=0x0C
ADD R6, R0, #10	rom: 0x0C	PC=0x10
CMP R6, #10	rom: 0x10	PC=0x14
BEQ pula1	rom: 0x14	PC=0x20
SUB R0, R1, #3	rom: 0x18	
ADD R0, R2, R5	rom: 0x1C	

pula1

MOV R3, #3	rom: 0x20	PC=0x24
MOV R2, #2	rom: 0x24	PC=0x28



Cortex-M ISA

Saltos - Branches



- Assembly

- Referências

Start

MOV R0, #4	rom: 0x08	PC=0x0C	
ADD R6, R0, #10	rom: 0x0C	PC=0x10	
LSL R2, R6, #2	rom: 0x10	PC=0x14	
BL pula1	rom: 0x14	PC=0x20	LR=0x18
SUB R0, R1, #3	rom: 0x18		
ADD R0, R2, R5	rom: 0x1C		

pula1

MOV R3, #3	rom: 0x20	PC=0x24
MOV R2, #2	rom: 0x24	PC=0x28
BX LR	rom: 0x28	



- Assembly

- Referências

Cortex-M ISA

Saltos - Branches

Start

MOV R0, #4	rom: 0x08	PC=0x0C	
ADD R6, R0, #10	rom: 0x0C	PC=0x10	
LSL R2, R6, #2	rom: 0x10	PC=0x14	
BL pula1	rom: 0x14	PC=0x20	LR=0x18
SUB R0, R1, #3	rom: 0x18		
ADD R0, R2, R5	rom: 0x1C		

pula1

MOV R3, #3	rom: 0x20	PC=0x24	
BL func2	rom: 0x24	PC=0x30	LR=0x28
MOV R2, #2	rom: 0x28	PC=0x34	
BX LR	rom: 0x2C	PC=LR=0x28	

func2

SUB R1, #3	rom: 0x30	PC=0x34	
BX LR	rom: 0x34	PC=LR=0x28	



- Assembly

- Referências

Cortex-M ISA

Saltos - Branches

Start

MOV R0, #4	rom: 0x08	PC=0x0C	
ADD R6, R0, #10	rom: 0x0C	PC=0x10	
LSL R2, R6, #2	rom: 0x10	PC=0x14	
BL pula1	rom: 0x14	PC=0x20	LR=0x18
SUB R0, R1, #3	rom: 0x18		
ADD R0, R2, R5	rom: 0x1C		

pula1

MOV R3, #3	rom: 0x20	PC=0x24	
PUSH {LR}	rom: 0x24	PC=0x28	
BL func2	rom: 0x28	PC=0x38	LR=0x2C
POP {LR}	rom: 0x2C	PC=0x30	LR=0x18
MOV R2, #2	rom: 0x30	PC=0x34	
BX LR	rom: 0x34	PC=LR=0x18	

func2

SUB R1, #3	rom: 0x38	PC=0x34	
BX LR	rom: 0x3C	PC=LR=0x28	



Cortex-M ISA

Exercício

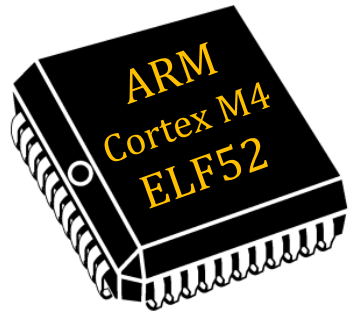
7. Faça um código que realize os seguintes passos e depois depure no Keil: *(Acrescentar ao final do arquivo a instrução **NOP**)*

- a) Mover para o **R0** o valor **10**;
- b) Somar **R0** com **5** e colocar o resultado em **R0**;
- c) Enquanto a resposta não for **50** somar mais **5**;
- d) Quando a resposta for **50** chamar uma função que:
 - d.1) Copia o **R0** para **R1**;
 - d.2) Verifica se **R1** é menor que **50**;
 - d.3) Se for menor que **50** incrementa, caso contrário modifica para **-50**
- e) Depois que retornar da função coloque uma instrução **NOP**
- f) Acrescente uma instrução para ficar travado na última linha de execução.



- Assembly

- Referências



- Assembly

- Referências

Cortex-M

Diretivas do Keil

- Auxiliam o processo de montagem (*assembly*).
- Não fazem parte do conjunto de instruções.
 - **CODE:** espaço para instruções de máquina (ROM)
 - **DATA:** espaço para variáveis globais (RAM)
 - **STACK:** espaço para pilha (RAM)
 - **ALIGN=n:** começa a área alinhada para 2ⁿ bytes
 - **|.text|:** seções de código produzidas pelo compilador C. Faz o código *assembly* poder ser chamado do C
 - **NOINIT:** faz uma área da RAM ser não inicializada



- Assembly

- Referências

Cortex-M

Diretivas do Keil

```
AREA RESET, CODE, READONLY           ;reset vectors in flash ROM
AREA DATA                            ;places objects in data memory (RAM)
AREA |.text|, CODE, READONLY, ALIGN=2 ;code in flash ROM
AREA STACK, NOINIT, READWRITE, ALIGN=3 ;stack área
```

- Para *linkar* variáveis e funções entre dois arquivos:
 - **EXPORT** ou **GLOBAL**: no arquivo onde o objeto está definido
 - **IMPORT** ou **EXTERN**: no arquivo que está tentando acessar

```
IMPORT name                           ;imports function "name" from other file
EXPORT name                           ;exports public function "name" for use
                                     ;elsewhere
```



- Assembly

- Referências

Cortex-M

Diretivas do Keil

- **ALIGN:** garante que o próximo objeto será alinhado propriamente. Recomendável colocar ao final do arquivo.

```
ALIGN          ;skips 0 to 3 bytes to make next word aligned
ALIGN 2        ;skips 0 or 1 byte to make next halfword aligned
ALIGN 4        ;skips 0 to 3 bytes to make next word aligned
```

- **THUMB:** colocada no topo do arquivo para especificar que o código será gerado com instruções Thumb.

```
THUMB
```

- **END:** Diretiva no fim de cada arquivo

```
END
```



- Assembly

- Referências

Cortex-M

Diretivas do Keil

- **Adicionando variáveis e constantes:**

DCB expr{,expr}	;places 8-bit byte(s) into memory
DCW expr{,expr}	;places 16-bit halfword(s) into memory
DCD expr{,expr}	;places 32-bit word(s) into memory
SPACE size	;reserves size bytes, uninitialized

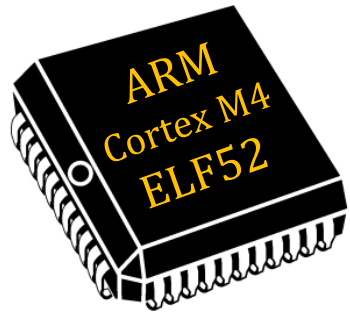
- **Como fazer um vetor?**

Declarar na região da RAM (abaixo do AREA DATA)

```
nome_do_vetor    SPACE    5 (5 bytes)
```

Na região do código quando for utilizar

```
LDR R0, =nome_do_vetor    (região inicial, diretiva Keil)
LDRB R1, [R0]              (vetor de bytes)
```



Cortex-M

Diretivas do Keil



- Assembly

- Referências

- **EQU** define um nome simbólico à uma constante numérica.
- Exemplos:

```
GPIO_PORTD_DATA_R EQU 0x400073FC  
GPIO_PORTD_DIR_R   EQU 0x40007400  
GPIO_PORTD_DEN_R   EQU 0x4000751C
```



Cortex-M ISA

Exercício

8. Ler de uma posição da memória **RAM** um número e calcular o fatorial. Armazenar o resultado em **R0**.



- Assembly

- Referências



Referências:



- Assembly

- Referências

Atividade Prática 1:

Adaptado por Prof. DaLuz

http://www.elf52.daeln.com.br/Labs/Laboratorio_AP1.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*