ELTD03z
Microcontroladores/Microprocessadores

Teoria_06a3

Prof. Enio R. Ribeiro

Universidade Federal de Itajubá - UNIFEI

T7.3) INSTRUÇÕES DE COMPARAÇÃO

Instruction: **CMP and CMN - (**CMP -> compare and CMN -> compare negative).

Syntax

```
CMP{cond} Rn, Operand2
CMN{cond} Rn, Operand2
```

- '*cond* ' is an optiional conditional code;
- '*Rn* ' is the register holding the first operand;
- '*Operand2* ' is a flexible second operand (constant or a register).

These instructions compare the value in a register with *operand2*. They update the condition flags on the result, but do not write the result to a register. **These instructions update the N, Z, C and V flags according to the result.**

The CMP instruction subtracts the value of *operand2* from the value in *Rn*. **This is the same as a SUBS instruction, except that the result is discarded.**

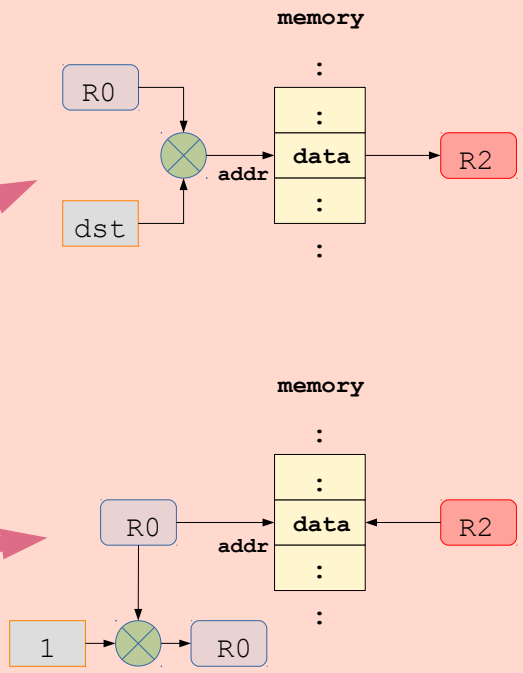
The CMN instruction adds the value of *operand2* to the value in *Rn*. **This is the same as an ADDS instruction, except that the result is discarded.**

```
    CMP     R2, R9          ;R2-R9
    CMN     R0, #6400       ;R0+#6400
```

T7.3a) INSTRUCÕES DE COMPARACÃO

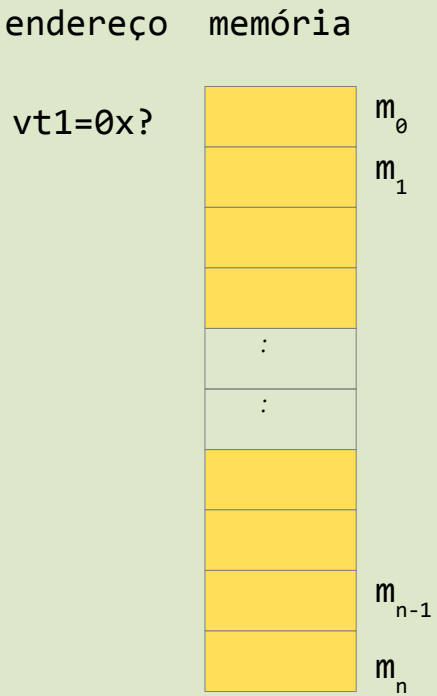
Ex. 3a – O vetor vt1 tem 10 bytes. Faça um programa para copiar os bytes de vt1, menores do que &50, no vetor vt2. O programa é cíclico. FDAN. O cálculo do offset entre os vetores deve ser dinâmico. Use post-indexed offset addressing.

```
;---Ex. 3a - End. index.; comparação
    export   __main
;===== diretiva area - dados (sram)
    area     dds1, data, readwrite
vt1 space    10
vtw dcb      &c7,&3b,&34,&af,&dd
vt2 space    10
;=== diretiva area - prog. (flash)
    area     m_prog, code, readonly
__main
    ldr      r0,=vt2        ;load pointer vt2
    mov      r1,#10         ;counter
pk0 ldrb     r2,[r0,#vt1-vt2] ;ler vt1(i)
    cmp      r2,#&50        ;compara(r2-&50)
    bhs      pk1            ;>&4f, não salva
    strb     r2,[r0],#1   ;salvar em vt2(i)
    b        pk2
pk1 add      r0,#1
pk2 subs     r1,#1          ;decr.counter
    bne pk0                 ;prox.elem.
    b   __main              ;
    end
```



7.3b. Exercícios: loop control -> counter

Ex_7.0a) Escreva um programa para inicializar os bytes do vetor vt1, o qual tem nb bytes. Condições: nb=8; usar instruções endereçamento indexado com auto pré/pós incr./decr.; programa cíclico; FDAN; usar um ponteiro. Método de controle de loop → contador.

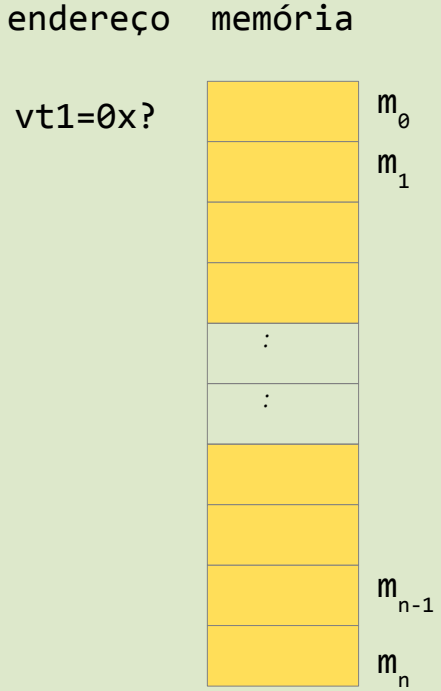



```
;---Ex_7.0a - inicializar bytes de vt
;controle->contador auxiliar
    export  __main
nb  equ     8
;===== diretiva area - dados (sram)
    area    dds1, data, readwrite
vt  space   nb      ;vetor origem
;=== diretiva area - prog. (flash)
    area    m_prog, code, readonly
__main
    ldr     r0,=vt  ;r0=&vt
    mov     r1,#&aa ;r1=&
    mov     r2,#nb  ;counter
pk0 strb    r1,[r0],#1;idx pos-inc
    subs    r2,#1   ;dec.counter
    bne     pk0 ;Z=0?(not zero?)
    b       __main  ;prox.elem.
    end
```

```
                __main:
0x0800024C 4804      LDR     r0,[pc,#16]  ;@0x08000260
0x0800024E F04F01AA  MOV     r1,#0xAA
0x08000252 F04F0208  MOV     r2,#0x08
0x08000256 F8001B01  STRB    r1,[r0],#0x01
0x0800025A 3A01      SUBS    r2,r2,#0x01
0x0800025C D1FB      BNE     0x08000256
0x0800025E E7F5      B       0x0800024C __main
```

7.3b Exercícios: loop control -> pointer (1)

Ex_7.0b) Escreva um programa para inicializar os bytes do vetor vt1, o qual tem nb bytes. Condições: nb=12; usar instruções endereçamento indexado com auto pré/pós incr./decr.; programa cíclico; FDAN; usar um ponteiro. Método de controle de loop → o ponteiro (1).

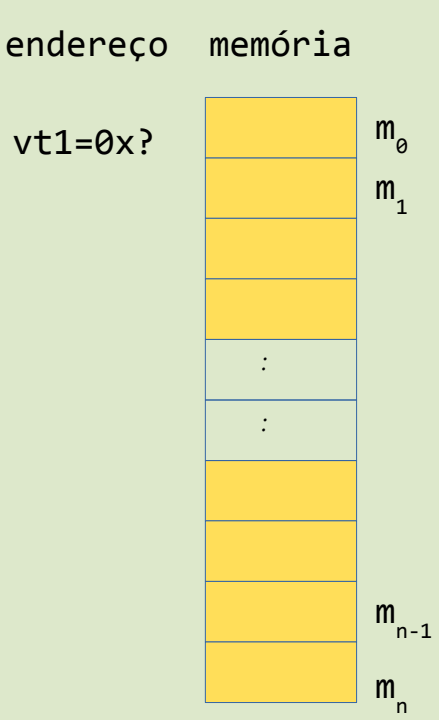



```
;---Ex_7.0b - inicializar bytes de vt
;controle -> ponteiro (1)
    export  __main
nb  equ     8
;===== diretiva area - dados (sram)
    area    dds1, data, readwrite
vt  space   nb  ;vetor origem
;=== diretiva area - prog. (flash)
    area    m_prog, code, readonly
__main
    ldr     r0,=vt  ;r0=&vt
    mov     r1,#&c9 ;r1=&c9
    mov     r2,r0   ;r2=r0
    add     r2,#nb  ;r2=r2+#nb
pk0 strb    r1,[r0],#1;idx pos-inc
    cmp     r0,r2   ;r0=r2?
    bne     pk0     ;Z=0?(not zero)
    b       __main  ;prox.elem.
    end
```

```
                __main:
0x0800024C 4805      LDR      r0,[pc,#20]   ;@0x08000264
0x0800024E F04F01C9  MOV      r1,#0xC9
0x08000252 4602      MOV      r2,r0
0x08000254 F1020208  ADD      r2,r2,#0x08
0x08000258 F8001B01  STRB     r1,[r0],#0x01
0x0800025C 4290      CMP      r0,r2
0x0800025E D1FB      BNE      0x08000258
0x08000260 E7F4      B        0x0800024C __main
```

7.3b Exercícios: loop control -> pointer (2)

Ex_7.0c) Escreva um programa para inicializar os bytes do vetor vt1, o qual tem nb bytes. Condições: nb=12; usar instruções endereçamento indexado com auto pré/pós incr./decr.; programa cíclico; FDAN; usar um ponteiro. Método de controle de loop → o ponteiro (2).




```
;---Ex_6.0c - inicializar bytes de vt
;controle -> ponteiro (2)
    export  __main
nb  equ     8
;===== diretiva area - dados (sram)
    area    dds1, data, readwrite
vt  space   nb  ;vetor origem
vtf             ;em branco
;vtf    dcb 0
;=== diretiva area - prog. (flash)
    area    m_prog, code, readonly
__main
    ldr     r0,=vt  ;r0=&vt
    mov     r1,#&7a ;r1=&7a
    ldr     r2,=vtf ;r2=&vtf
pk0 strb    r1,[r0],#1;idx pos-inc
    cmp     r0,r2   ;r0=r2?
    bne     pk0     ;Z=0?(not zero)
    b       __main  ;prox.elem.
    end
```

```
                __main:
0x0800024C 4804      LDR     r0,[pc,#16]  ;@0x08000260
0x0800024E F04F017A  MOV     r1,#0x7A
0x08000252 4A04      LDR     r2,[pc,#16]  ;@0x08000264
0x08000254 F8001B01  STRB    r1,[r0],#0x01
0x08000258 4290      CMP     r0,r2
0x0800025A D1FB      BNE     0x08000254
0x0800025C E7F6      B       0x0800024C __main
```

T7.1) Instruções: multiplicação e divisão

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: **cd00228163-stm32f10xxx-cortex-m3-programming-manual-stmicroelectronics.pdf**

T7.1a) Instruções: multiplicação – 32-bit result

MUL, MLA, and MLS

Multiply, multiply with accumulate, and multiply with subtract, using 32-bit operands, and producing a 32-bit result.

Syntax

```
MUL{S}{cond} {Rd,} Rn, Rm ; Multiply
MLA{cond} Rd, Rn, Rm, Ra ; Multiply with accumulate
MLS{cond} Rd, Rn, Rm, Ra ; Multiply with subtract
```

where:

• '*cond*' is an optional condition code (see *Conditional execution on page 56*)

• '*S*' is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation (see *Conditional execution on page 56*).

• '*Rd*' is the destination register. If *Rd* is omitted, the destination register is *Rn*

• '*Rn*', '*Rm*' are registers holding the values to be multiplied

• '*Ra*' is a register holding the value to be added to or subtracted from

Fig. – Quadro: **cd00228163-stm32f10xxx-cortex-m3-programming-manual-stmicroelectronics.pdf**

T7.1a) Instruções: multiplicação – 32-bit result

Operation

The MUL instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The MLA instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.
The MLS instruction multiplies the values from *Rn* and *Rm*, subtracts the product from the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.
The results of these instructions do not depend on whether the operands are signed or unsigned.

Restrictions
In these instructions, do not use SP and do not use PC.
If you use the S suffix with the MUL instruction:
• *Rd*, *Rn*, and *Rm* must all be in the range R0 to R7
• *Rd* must be the same as *Rm*
• You must not use the *cond* suffix

Condition flags
If S is specified, the MUL instruction:
• Updates the N and Z flags according to the result
• Does not affect the C and V flags

Quadro: **cd00228163-stm32f10xxx-cortex-m3-programming-manual-stmicroelectronics.pdf**

T7.1a) Instruções: multiplicação “16x16 bits”

```
;---Ex-1a - multiply instructions
    export  __main
;---directive area-programa
    area    m_prog, code, readonly
__main
    movw    r0,#65535   ;r0=&ffff
    movw    r1,#65535   ;r1=&ffff
    mul     r2,r0,r1    ;R2=r0*r1=&fffe0001
    movw    r3,#0       ;r3=65536
    movt    r3,#1       ;r3=&10000
    movw    r4,#65535   ;r4=&ffff
    mul     r5,r3,r4    ;R5=&ffff0000
    mov     r6,r3       ;r6=r3=&10000
    mov     r7,r3       ;r7=r3=&10000
    mul     r8,r6,r7    ;r8=&00000000
pk0 b       pk0
    end
```

```
                    __main:
0x0800024C F64F70FF  MOVW     r0,#0xFFFF
0x08000250 F64F71FF  MOVW     r1,#0xFFFF
0x08000254 FB00F201  MUL      r2,r0,r1
0x08000258 F2400300  MOVW     r3,#0x00
0x0800025C F2C00301  MOVT     r3,#0x01
0x08000260 F64F74FF  MOVW     r4,#0xFFFF
0x08000264 FB03F504  MUL      r5,r3,r4
0x08000268 461E      MOV      r6,r3
0x0800026A 461F      MOV      r7,r3
0x0800026C FB06F807  MUL      r8,r6,r7
0x08000270 E7FE      B        0x08000270
```

T7.1b) Instruções: multiplicação 32x32 bits

UMULL, UMLAL, SMULL, and SMLAL

Signed and unsigned long multiply, with optional accumulate, using 32-bit operands and producing a 64-bit result.

Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

• ‘op’ is one of:

UMULL: Unsigned long multiply

UMLAL: Unsigned long multiply, with accumulate

SMULL: Signed long multiply

SMLAL: Signed long multiply, with accumulate

• ‘cond’ is an optional condition code (see Conditional execution on page 56)

• ‘RdHi, RdLo’ are the destination registers. For UMLAL and SMLAL, they also hold the accumulating value.

• ‘Rn, Rm’ are registers holding the operands

Quadro: **cd00228163-stm32f10xxx-cortex-m3-programming-manual-stmicroelectronics.pdf**

T7.1b) Instruções: multiplicação 32x32 bits

Operation

The UMULL instruction interprets the values from Rn and Rm as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in RdLo, and the most significant 32 bits of the result in RdHi.

The UMLAL instruction interprets the values from Rn and Rm as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in RdHi and RdLo, and writes the result back to RdHi and RdLo.
The SMULL instruction interprets the values from Rn and Rm as two's complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in RdLo, and the most significant 32 bits of the result in RdHi.
The SMLAL instruction interprets the values from Rn and Rm as two's complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in RdHi and RdLo, and writes the result back to RdHi and RdLo.

Restrictions
In these instructions:
• Do not use either SP or PC
• RdHi and RdLo must be different registers

Condition flags
These instructions do not affect the condition code flags.

Quadro: **cd00228163-stm32f10xxx-cortex-m3-programming-manual-stmicroelectronics.pdf**

T7.1b) Instruções: multiplicação “32x32 bits”

```
;---Ex-1b - multiply instructions 32x32 bits
    export  __main
;---directive area-programa
    area    m_prog, code, readonly
__main
    movw    r0,#65535   ;r0Lo=&ffff
    movt    r0,#65535   ;r0Hi=&ffff
    mov     r1,r0       ;r1=r0
    umull   r2,r3,r0,r1 ;r3Hi:r2Lo=r0*r1
                        ;=&fffffffe0000001
pk0 b   pk0
    end
```

```
                    __main:
0x0800024C F64F70FF  MOVW     r0,#0xFFFF
0x08000250 F6CF70FF  MOVT     r0,#0xFFFF
0x08000254 4601      MOV      r1,r0
0x08000256 FBA02301  UMULL    r2,r3,r0,r1
0x0800025A E7FE      B        0x0800025A
```

T7.1c) Multiplicação - exercícios

Ex_1c – Seja vt1 um vetor com 8 fatores e cada fator de 1 byte. Faça um programa para multiplicar cada byte de vt1 pela variável vk, de 1 byte, e guarde resultado em vtk. O vetor vt1 e vk devem estar memória flash; o vetor vtk em memória sram. FDAN. Programa cíclico. Usar ponteiros com offset fixo.

```
;---Ex_1c - mult. vetor*vk
    export   __main
;---diretiva area-dados1
    area     d_1, data, readwrite
vt1 dcb  &aa,&39,&ca,&f3,&e5,&99,&fe,&c9
vk  dcb  &d9
;---diretiva area-dados2 -sram
    area     d_2, data, readwrite
vtk space    16
;---directive area-programa
    area     m_prog, code, readonly
__main
pk0 ldr      r0,=vt1  ;r0=&vt1
    ldr      r1,=vtk  ;r1=&vtk
    mov      r2,#8    ;cont.
    ldrb     r3,[r0,#vk-vt1];r3=vk
pk1 ldrb     r4,[r0]  ;r4=vt(i)
    mul      r5,r3,r4
    strh     r5,[r1]
    add      r0,#1
    add      r1,#2
    subs     r2,#1
    bne      pk1  ;
    b        pk0  ;
    end
```

```
                    __main:
0x0800024C 4807      LDR      r0,[pc,#28]
0x0800024E 4908      LDR      r1,[pc,#32]
0x08000250 F04F0208  MOV      r2,#0x08
0x08000254 7A03      LDRB     r3,[r0,#0x08]
0x08000256 7804      LDRB     r4,[r0,#0x00]
0x08000258 FB03F504  MUL      r5,r3,r4
0x0800025C 800D      STRH     r5,[r1,#0x00]
0x0800025E F1000001  ADD      r0,r0,#0x01
0x08000262 F1010102  ADD      r1,r1,#0x02
0x08000266 3A01      SUBS     r2,r2,#0x01
0x08000268 D1F5      BNE      0x08000256
0x0800026A E7EF      B        0x0800024C __main
```

T7.1c) Multiplicação - exercícios

Ex_1d – Refaça o exercício Ex_1c, usando ponteiros com auto pré/pós incremento/decremento. Elimine o contador auxiliar. Faça o controle de loop por um dos ponteiros.

Ex_1e – Sejam vt1 e vt2 com nb fatores e cada fator é uma halfword. Faça um programa para multiplicar vt1 por vt2 e guarde o resultado em vtk. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico.

T7.2a) Instruções: divisão

SDIV and UDIV
Signed divide and unsigned divide.

Syntax

```
SDIV{cond} {Rd,} Rn, Rm
UDIV{cond} {Rd,} Rn, Rm
```

where:

• '*cond*' is an optional condition code (see *Conditional execution on page 56*)

• '*Rd*' is the destination register (quociente). If *Rd* is omitted, the destination register is *Rn*

• '*Rn,*' is the register holding the value to be divided (dividendo/numerador)

• '*Rm*' is a register holding the divisor (divisor/denominador)

T7.2a) Instruções: divisão

Operation

SDIV performs a signed integer division of the value in *Rn* by the value in *Rm*.

UDIV performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

Restrictions
Do not use either SP or PC.

Condition flags
These instructions do not change the flags.

T7.2b) Instruções: divisão – binário puro

```
;---Ex-2a - divide unsigned bin 32/32 bits
    export  __main
;---directive area-programa
    area    m_prog, code, readonly
__main
    movw    r0,#65534   ;r0=&fffe
    movt    r0,#65535   ;r0=&ffff
    movw    r1,#2
    udiv    r2,r0,r1    ;r2=r0/r1
pk0 b   pk0
    end
```

```
                    __main:
0x0800024C F64F70FE  MOVW     r0,#0xFFFE
0x08000250 F6CF70FF  MOVT     r0,#0xFFFF
0x08000254 F2400102  MOVW     r1,#0x02
0x08000258 FBB0F2F1  UDIV     r2,r0,r1
0x0800025C E7FE      B        0x0800025C
```

T7.2c) Instruções: divisão – binário sinalizado

```
;---Ex-2b - divide signed bin 32/32 bits
    export  __main
;---directive area-programa
    area    m_prog, code, readonly
__main
    movw    r0,#65534   ;r0=&fffe
    movt    r0,#32767   ;r0=&7fff
    mov     r1,#2
    sdiv    r2,r0,r1    ;r2=r0/r1=&3fffffff
    movw    r3,#&0002   ;
    movt    r3,#&8000   ;
    sdiv    r4,r3,r1    ;r4=r3/r1=&c0000001
    mvn     r5,r1       ;r5=1's de r1
    add     r5,#1       ;r5=2's de r1
    sdiv    r6,r3,r5    ;r6=r3/r5=&ffffffff
pk0 b       pk0
    end
```

```
                    __main:
0x0800024C F64F70FE  MOVW     r0,#0xFFFE
0x08000250 F6C770FF  MOVT     r0,#0x7FFF
0x08000254 F04F0102  MOV      r1,#0x02
0x08000258 FB90F2F1  SDIV     r2,r0,r1
0x0800025C F2400302  MOVW     r3,#0x02
0x08000260 F2C80300  MOVT     r3,#0x8000
0x08000264 FB93F4F1  SDIV     r4,r3,r1
0x08000268 EA6F0501  MVN      r5,r1
0x0800026C F1050501  ADD      r5,r5,#0x01
0x08000270 FB93F6F5  SDIV     r6,r3,r5
0x08000274 E7FE      B        0x08000274
```

T7.2d) Divisão – exercícios bin. não sinalizados

```
;---Ex_2c - div. b/b (32/32 bits)
    export  __main
;---diret. equate
nb  equ 8
;---diretiva area-dados1
    area    d_1, data, readwrite
vr  space   nb
vs  space   nb
;---diretiva area-dados2 -sram
    area    d_2, data, readwrite
vrs space   nb
;---directive area-programa
    area    m_prog, code, readonly
__main
pk0 ldr     r0,=vr  ;r0=&vr
    ldr     r1,=vrs ;r1=&vrs
    ldr     r2,=vr+nb   ;r2=&vs(contr.)
pk1 ldrb    r4,[r0,#vs-vr];r4=vs(i)(dvs)
    ldrb    r3,[r0],#1 ;r3=vr(i)(ddo)
    udiv    r5,r3,r4    ;r5=r3/r4
    strb    r5,[r1],#1 ;
    cmp     r0,r2   ;r0=r2?(r0-r2)
    bne     pk1     ;
    b       pk0     ;
    end
```

Ex_2c – Sejam vr e vs com nb bytes, cujos valores são binários não sinalizados. Faça um programa para dividir vr por vs e guarde o resultado em vrs. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico. Condição: nb=8.

```
                    __main:
0x0800024C 4806      LDR      r0,[pc,#24]
0x0800024E 4907      LDR      r1,[pc,#28]
0x08000250 4A07      LDR      r2,[pc,#28]
0x08000252 7A04      LDRB     r4,[r0,#0x08]
0x08000254 F8103B01  LDRB     r3,[r0],#0x01
0x08000258 FBB3F5F4  UDIV     r5,r3,r4
0x0800025C F8015B01  STRB     r5,[r1],#0x01
0x08000260 4290      CMP      r0,r2
0x08000262 D1F6      BNE      0x08000252
0x08000264 E7F2      B        0x0800024C
```

T7.2d) Divisão – exercícios bin. não sinalizados

Ex_2d – Sejam vr e vs com nb halfwords, cujos valores são binários não sinalizados. Faça um programa para dividir vr por vs e guarde o resultado em vrs. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico. Condição: nb=8.

Ex_2e – Sejam vr e vs com nb words, cujos valores são binários não sinalizados. Faça um programa para dividir vr por vs e guarde o resultado em vrs. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico. Condição: nb=4.

T7.2d) Divisão – exercícios bin. sinalizado

```
;---Ex_2f - div. b/b sinal.
    export  __main
;---diret. equate
nb  equ 8
;---diretiva area-dados1
    area    d_1, data, readwrite
vr  space   nb
vs  space   nb
;---diretiva area-dados2 -sram
    area    d_2, data, readwrite
vrs space   nb
;---directive area-programa
    area    m_prog, code, readonly
__main
pk0 ldr     r0,=vr  ;r0=&vr
    ldr     r1,=vrs ;r1=&vrs
;   ldr     r2,=vr+nb ;r2=&vs(contr.)
    mov     r2,r0   ;
    add     r2,#nb  ;
pk1 ldrsb   r4,[r0,#vs-vr];r4=vs(i)(dvs)
    ldrsb   r3,[r0],#1 ;r3=vr(i)(ddo)
    sdiv    r5,r3,r4    ;r5=r3/r4
    strb    r5,[r1],#1
    cmp     r0,r2   ;ro=r2?(r0-r2)
    bne     pk1 ;
    b       pk0 ;
    end
```

Ex_2f – Sejam vr e vs com nb bytes, cujos valores são binários sinalizados. Faça um programa para dividir vr por vs e guarde o resultado em vrs. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico. Condição: nb=8.

T7.2d) Divisão – exercícios bin. sinalizado

Ex_2g – Sejam vr e vs com nb halfwords, cujos valores são binários sinalizados. Faça um programa para dividir vr por vs e guarde o resultado em vrs. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico. Condição: nb=8.

Ex_2h – Sejam vr e vs com nb words, cujos valores são binários não sinalizados. Faça um programa para dividir vr por vs e guarde o resultado em vrs. Todos os vetores devem estar memória sram. Use ponteiros com auto pré/pós incremento/decremento. Faça o controle de loop por ponteiro. FDAN. Programa cíclico. Condição: nb=4.

```
                __main:
0x080002BC 4807      LDR      r0,[pc,#28]  ; @0x080002DC
0x080002BE 4908      LDR      r1,[pc,#32]  ; @0x080002E0
0x080002C0 4602      MOV      r2,r0
0x080002C2 F1020208  ADD      r2,r2,#0x08
0x080002C6 F9904008  LDRSB    r4,[r0,#0x08]
0x080002CA F9103B01  LDRSB    r3,[r0],#0x01
0x080002CE FB93F5F4  SDIV     r5,r3,r4
0x080002D2 F8015B01  STRB     r5,[r1],#0x01
0x080002D6 4290      CMP      r0,r2
0x080002D8 D1F5      BNE      0x080002C6
0x080002DA E7EF      B        0x080002BC __main
```

Programa assemblado referente ao programa fonte Ex_2f.

T7.4) Cálculo de tempo de programa

```
;---Ex-3a - cálculo tempo programa
    export  __main
;---diretiva area-dados
    area    d_1, data, readonly
vt1 dcb     &aa,&39,&ca,&f3,&e5
;---directive area-programa
    area    m_prog, code, readonly
__main
pk0 ldr     r0,=vt1 ;       <- a
    mov     r1,#4   ;       <- b
    mov     r3,#0   ;       <- c
pk1 ldrb    r2,[r0] ;       <- d
    add     r3,r2   ;       <- e
    subs    r1,#1   ;       <- f
    bne pk1         ;       <- g/G
    b   pk0         ;       <- h
    end
```

Ex_3b) Expressar algebricamente o número de ciclos para a execução do programa dado.

Para o anel:

$N1 = \beta(d+e+f)+(\beta-1)g+G$
$N1 = \beta(d+e+f+g)-g+G$

$\beta = 4$

Para o programa completo:
$Nc = a+b+c+N1+h$

$Nc = a+b+c+4(d+e+f+g)-g+G+h$

```
                __main:
0x0800024C 4804      LDR      r0,[pc,#16]  ;@0x08000260
0x0800024E F04F0104  MOV      r1,#0x04
0x08000252 F04F0300  MOV      r3,#0x00
0x08000256 7802      LDRB     r2,[r0,#0x00]
0x08000258 4413      ADD      r3,r3,r2
0x0800025A 3901      SUBS     r1,r1,#0x01
0x0800025C D1FB      BNE      0x08000256
0x0800025E E7F5      B        0x0800024C
```

Ex_3c) Sabendo que um ciclo de MPU é de 0,125 us calcule o tempo gasto para executar o programa dado. STM32F103 –> f=8 MHz.

Solução: Atribuir às letras número de ciclos de cada instrução e substituir na expressão algébrica obtida.

Branch	Conditional	B<cc> <label>	1 or 1 + P[d]
	Unconditional	B <label>	1 + P
Move	Register	MOV Rd, <op2>	1
	16-bit immediate	MOVW Rd, #<imm>	1
	Byte	LDRB Rd, [Rn, <op2>]	2[c]

Instructions count cycles: **DDI0337H_cortex_m3_r2p0_trm.pdf – BAIXAR MANUAL!!!**
Veja a seção "3.3 Instruction set summary".

```
<- a
<- b
<- c
<- d
<- e
<- f
<- g
<- h
```

```
<- a=2
<- b=1
<- c=1
<- d=2
<- e=1
<- f=1
<- g/G=3/1
<- h=3
```

Nc= a+b+c+4(d+e+f+g)-g+G+h

T= Nc*Tciclo

T= tempo total programa
Tciclo = tempo de um ciclo
Tciclo=1/f=1/(8 MHz).

Nc= 2+1+1+4(2+1+1+3)-3+1+3
Nc= 5+4(7)=33

T=33x0,125 us => T= 4,125 us

T 7.4a) Cálculo de tempo de programa - exercícios

Ex_3d1 – Seja vr uma variável de 1 byte, não sinalizada e maior do que 1. Faça um programa para ler vr e executar determinado trecho de programa que tenha um número de ciclos de execução proporcional ao valor de vr. Admita que o STM32F103 possua somente registros de 1 byte. FDAN. Programa cíclico. Frequência do micro: 8 MHz.

```
;---Ex_3d1 - loop tempo
    export   __main
;---diret. area d_1 - sram
    area     d_1, data, readwrite
vr  dcb      2
;---directive area-programa
    area     m_prog, code, readonly
__main
    ldr      r0,=vr    ;r0=&vr
pk0 ldrb     r1,[r0]   ;r1=r0
pk1 nop                ;
    nop                ;
    nop                ;
    nop                ;
    subs     r1,#1     ;
    bne      pk1       ;
    b        pk0       ;
    end
```

Ex_3d1_a – Escreva a expressão algébrica para anel interno do programa dado em “Ex_3d1”.

Ex_3d1_b – Escreva a expressão algébrica para o programa dado em “Ex_3d1”.

Ex_3d1_c – O micro STM32F103 está funcionando com a frequência de 8 MHz. Para vr= 20, qual é o tempo consumido pelo anel interno? Qual é o tempo consumido pelo programa.

Ex_3d2 – Seja vr uma variável de 1 byte e sinalizada. Faça um programa para ler vr e executar determinado trecho de programa que: a) dure 400 µs, se vr positiva, b) dure 550 µs, se vr negativa. Admita que o STM32F103 possua somente registros de 1 byte. FDAN. Programa cíclico. Frequência do micro: 8 MHz.

Ex_3d3 – Seja vr uma variável de 1 byte e sinalizada. Faça um programa para ler vr e executar determinado trecho de programa que: a) dure 400 ms, se vr positiva, b) dure 275 ms, se vr negativa. Admita que o STM32F103 possua somente registros de 2 bytes. FDAN. Programa cíclico. Frequência do micro: 8 MHz.

Ex_3d4 – Seja vr uma variável de 1 byte, não sinalizada e 0<vr<65. Faça um programa para ler vr e executar determinado trecho de programa que consuma 1 s para cada unidade de vr, isto é, se vr = 5 então o trecho deve consumir 5 s; se vr = 9 então o trecho deve consumir 9 s, etc. Admita que o STM32F103 está funcionando com uma frequência de 8 MHz. FDAN. Programa cíclico.