ELTD03z
Microcontroladores/Microprocessadores

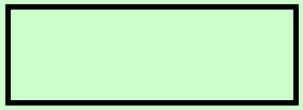
Teoria_04a1_1b

Prof. Enio R. Ribeiro

Universidade Federal de Itajubá - UNIFEI

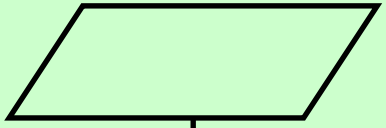
T4.1) Fluxogramas: elementos básicos

1) início, término

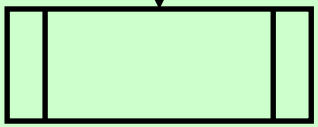


2) processo, comando, ação

3) entrada/saída

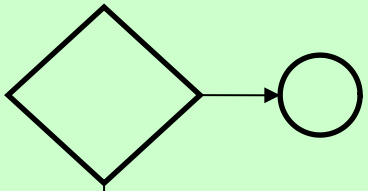


4) fluxo/conexão



8) sub-rotina

5) decisão

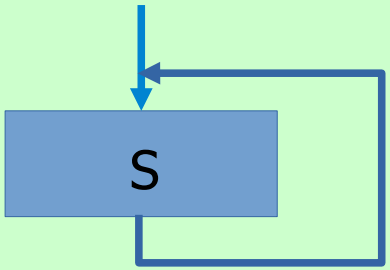


6) conexão local

7) conexão externa

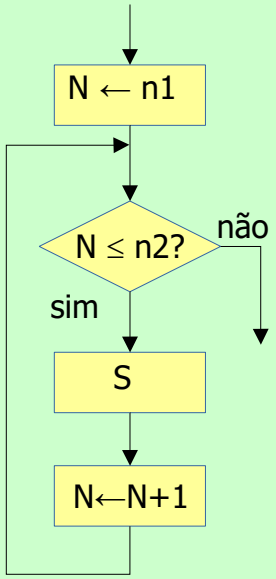
T4.2) Anéis (loops) de programa: fluxogramas básicos

T4.2a) Faça S para sempre (*DO statement S forever*)

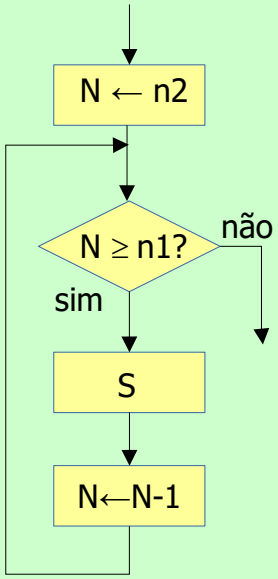



```
        :
        nop             ;*no operation
pk1     b       pk1     ;loop - branch incond.
        end
```

T4.2b) Para N= n1 até n2, faça S (n1≤n2) – (For N DO S)



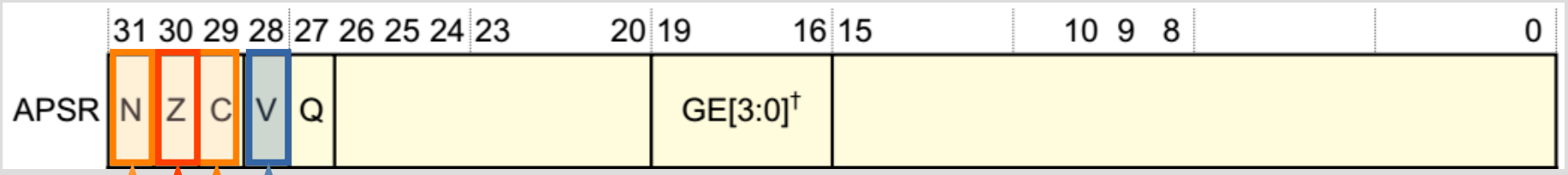
(a) para N, n1 → n2, faça S.



(b) para N, n2 → n1, faça S.

T4.3) REGISTRO : Current Program Status Register - (xPSR ~> APSR)

REGISTRO ESTADO (CONDIÇÃO!)



FLAG V – (Overflow): Overflow condition code flag. Set to 1 if the instruction results in an overflow condition, for example a signed overflow on an addition.

FLAG C – (Carry or borrow)): Carry condition code flag. Set to 1 if the instruction results in a carry condition, for example an unsigned overflow on an addition.

0: Add operation did not result in a carry bit or subtract operation resulted in a borrow bit;
1: Add operation resulted in a carry bit or subtract operation did not result in a borrow bit.

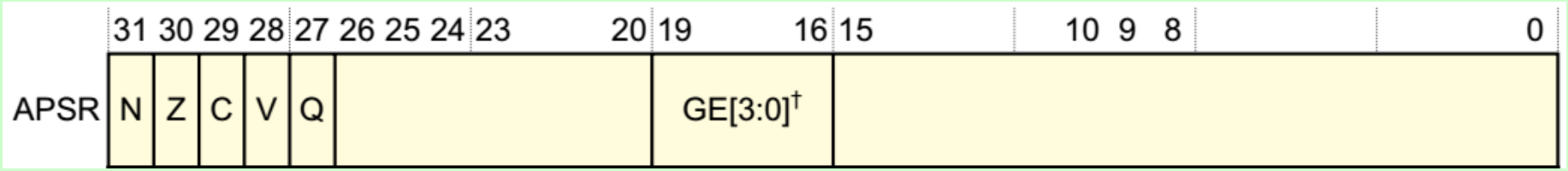
FLAG Z – (Zero): Zero condition code flag. Set to 1 if the result of the instruction is zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

FLAG N – (Negativo): Negative condition code flag. Set to bit[31] of the result of the instruction. If the result is regarded as a two's complement signed integer, then N = 1 if the result is negative and N = 0 if it is positive or zero.

0: Operation result was positive, zero, greater than, or equal;
1: Operation result was negative or less than.

These bits are essential for the use of conditional operations. Updating of these flags is carried out by most of the instructions, *provided that the suffix S is specified in the symbolic name of the instruction*.

T4.3a) REGISTRO : Current Program Status Register - APSR (xPSR)



N

1) Qual é o estado do flag N, após a execução de cada instrução dada.

```
    :
    ldr     r2,[r0]     ;(1)
    movs    r4,r2       ;(2)
    ldr     r3,[r1]     ;(3)
    movs    r5,r3       ;(4)
    :
```

Resposta: o flag terá o seguinte valor para cada instrução.

Em (1):

R2 = 7e00 0000; N=0 (não afetado)

Em (2):

R4 = 7e00 0000; N=0 (afetado)

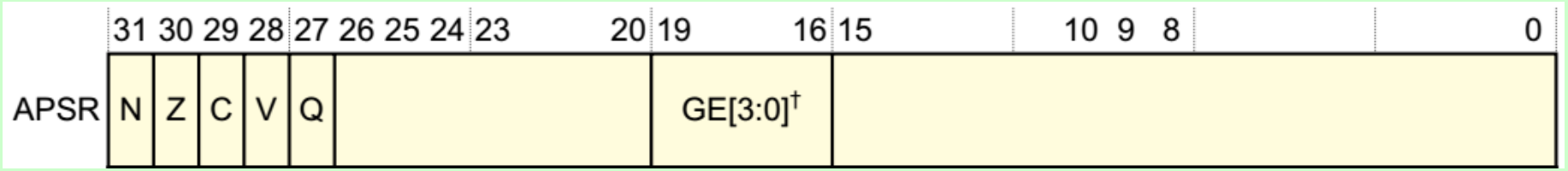
Em (3):

R3 = 8f00 0000; N=0 (não afetado)

Em (4):

R5 = 8f00 0000; N=1 (afetado)

T4.3b) REGISTRO : Current Program Status Register - APSR (xPSR)



Z

1) Qual é o estado do flag Z, após a execução de cada instrução dada.

```
    :
    ldr     r2,[r0]     ;(1)
    movs    r4,r2       ;(2)
    ldr     r3,[r1]     ;(3)
    movs    r5,r3       ;(4)
    :
```

Resposta: o flag terá o seguinte valor para cada instrução.

Em (1):

R2 = 7700 0000; Z=0 (Z não é afetado)

Em (2):

R4 = 7700 0000; Z=0 (Z é afetado)

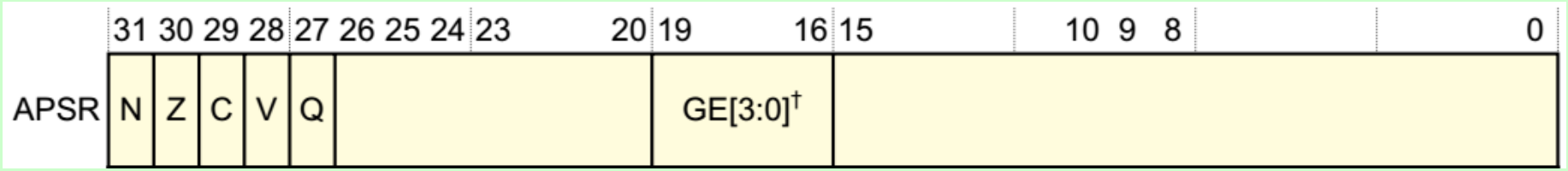
Em (3):

R3 = 0000 0000; Z=0 (Z não é afetado)

Em (4):

R5 = 0000 0000; Z=1 (Z é afetado)

T4.3c1) REGISTRO : Current Program Status Register - APSR (xPSR)



C

1) Qual é o estado do flag C, após a execução de cada instrução dada.

```
    :
    ldr     r2,[r0]     ;(1)
    ldr     r3,[r1]     ;(2)
    adds    r4,r3,r2    ;(3)
    adds    r5,r4,r2    ;(4)
    :
```

Resposta: o flag terá o seguinte valor para cada instrução.

Em (1):

R2 = 6000 0000; C=0 (C não é afetado)

Em (2):

R3 = 7000 0000; C=0 (C não é afetado)

Em (3):

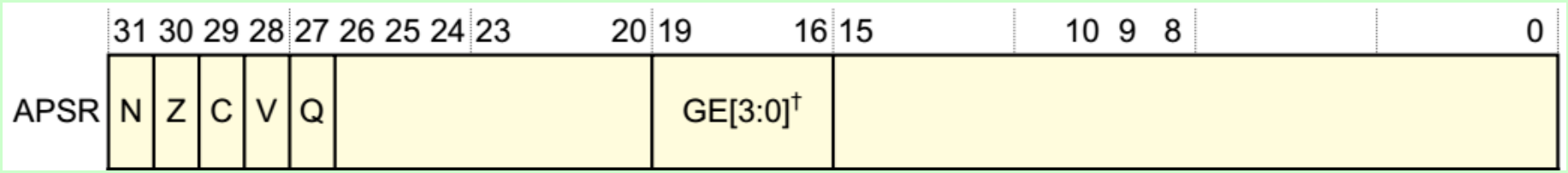
R4 = d000 0000; C=0 (C é afetado)

Em (4):

R5 = 3000 0000; C=1 (C é afetado)

OBS. OUTROS FLAGS PODEM SER AFETADOS!

T4.3c2) REGISTRO : Current Program Status Register - APSR (xPSR)



C

1) Qual é o estado do flag C, após a execução de cada instrução dada.

```
    :
    ldr     r2,[r0]     ;(1)
    ldr     r3,[r1]     ;(2)
    subs    r4,r2,r3    ;(3)
    subs    r5,r3,r2    ;(4)
    :
```

Resposta: o flag terá o seguinte valor para cada instrução.
Em (1):
R2 = 6000 0000; C=0 (C não é afetado)

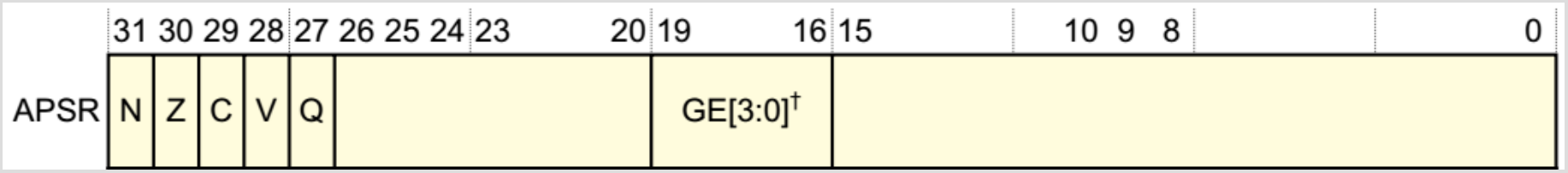
Em (2):
R3 = 7000 0000; C=0 (C não é afetado)

Em (3):
R4 = f000 0000; C=0 (C é afetado – borrow)

Em (4):
R5 = 1000 0000; C=1 (C é afetado – não borrow)

OBS. OUTROS FLAGS PODEM SER AFETADOS!

T4.4) BRANCH CONDICIONAL -> Current Program Status Register - APSR (xPSR)



BRANCH (DESVIO) CONDICIONAL
1) FAZ O TESTE DE UMA CONDIÇÃO
2) A CONDIÇÃO → REPRESENTADA → 1 OU MAIS BITS (FLAGS)

BRANCH (DESVIO)
1) OCORRE >> "teste" for verdadeiro
2) NÃO OCORRE >> "teste" for falso => executa próxima instrução

Branch instructions

B, BL, BX, and BLX

Syntax

`B{cond} label`

`cond` -> condition code suffix

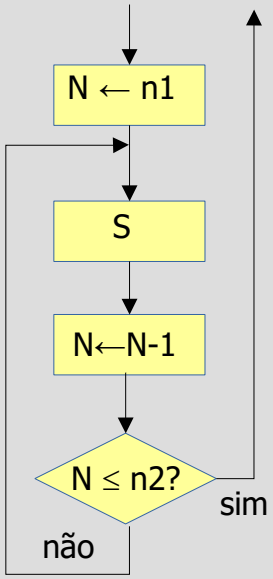
T4.5) CONDITION CODE SUFFIXES

Table 23. Condition code suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N ! = V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N ! = V	Less than or equal, signed ≤
AL	Can have any value	Always. This is the default when no suffix is specified.

Fig. 1 – Fonte da “Table 23”: **cd00228163-stm32f10xxx-cortex-m3-programming-manual-stmicroelectronics.pdf**

T4.6) BRANCH CONDICIONAL -> Exemplos



(a) para N, n1 → n2, faça S.

Ex. 1a - Faça um programa que implemente a ação indicada no fluxograma. (Uso flag Z).

Etapa 1 – Inicializar contador (registro);
Etapa 2 – executar S;
Etapa 3 – increm./decr. contador (registro);
Etapa 4 – testar contador

```
;---Ex. 1a - branch cond. (Z=0?)
    export   __main
;===============
;---diretiva area
    area     m_prog, code, readonly
__main
    mov      r0,#5     ;contador
pk1 nop      ;ação
    subs     r0,#1     ;atualizar r0 (decr.)
    bne      pk1       ;**branch condic.**(Z=0?)
    b        __main    ;reciclar -branch incond.
    end
```

Ex. 1a_1 – Transcreva o programa (Ex.1a) no ambiente Keil uVision e teste-o por completo.

Ex. 1a_2 – Refaça o (Ex.1a) considerando um valor entre &C0 e &DF para o contador. Incremente o contador. Teste-o por completo.

```
                    __main:
0x0800024C F04F0005  MOV       r0,#0x05
0x08000250 BF00      NOP
0x08000252 3801      SUBS      r0,r0,#0x01
0x08000254 D1FC      BNE       0x08000250
0x08000256 E7F9      B         0x0800024C __main
```

T4.6) BRANCH CONDICIONAL -> Exemplo

Ex. 1b - Faça um programa para ler a variável vr1: (a) se vr1 ≥ 0 , fazer o complemento de 1 de vr1; (b) se vr1< 0, fazer o complemento de 2 de vr1. (Uso flag N).

```
;---Ex. 1b - branch cond. (N=0?)
    export   __main
;=== diretiva area - dados (sram)
    area     dds1, data, readwrite
vr1 space    1
;=== diretiva area - prog. (flash)
    area     m_prog, code, readonly
__main
    ldr      r0,=vr1  ;carregar pointer
    ldrsb    r1,[r0]  ;ler vr1
    movs     r2,r1    ;
    bpl      pk1      ;**branch condicional(N=0?)**
pk2 mvn      r2,r1    ;
    add      r2,#1
    strb     r2,[r0]
    b        __main   ;
pk1 mvn      r2,r1    ;
    strb     r2,[r0]  ;
    b        __main   ;
    end
```

Instruction: **MOV / MVN**

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

The MVN instruction takes the value of *Operand2*, performs a bitwise logical NOT operation on the value and places the result into *Rd*.

Syntax

```
MOV{S}(cond) Rd, Operand2 ;(operand2)= Rm
MVN{S}(cond) Rd, Operand2 ;
```

- '*Rd* ' is the destination register
- '*cond* ' is an optional condition code
- '*Operand2* ' is a flexible second operand and it can be: constant; register with optional shift
- '*S*' is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation

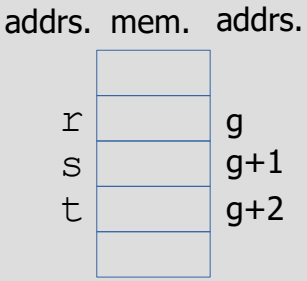
```
                    __main:
0x0800024C 4807      LDR      r0,[pc,#28]   ;
0x0800026C
0x0800024E F9901000  LDRSB    r1,[r0,#0x00]
0x08000252 000A      MOVS     r2,r1
0x08000254 D505      BPL      0x08000262
0x08000256 EA6F0201  MVN      r2,r1
0x0800025A F1020201  ADD      r2,r2,#0x01
0x0800025E 7002      STRB     r2,[r0,#0x00]
0x08000260 E7F4      B        0x0800024C __main
0x08000262 EA6F0201  MVN      r2,r1
0x08000266 7002      STRB     r2,[r0,#0x00]
0x08000268 E7F0      B        0x0800024C __main
```

Ex. 1b_1 - Faça um programa para ler a variável vr1: (a) se vr1 ≥ 0 , fazer o complemento de 2 de vr1; (b) se vr1< 0, fazer o complemento de 1 de vr1. (Uso flag N).

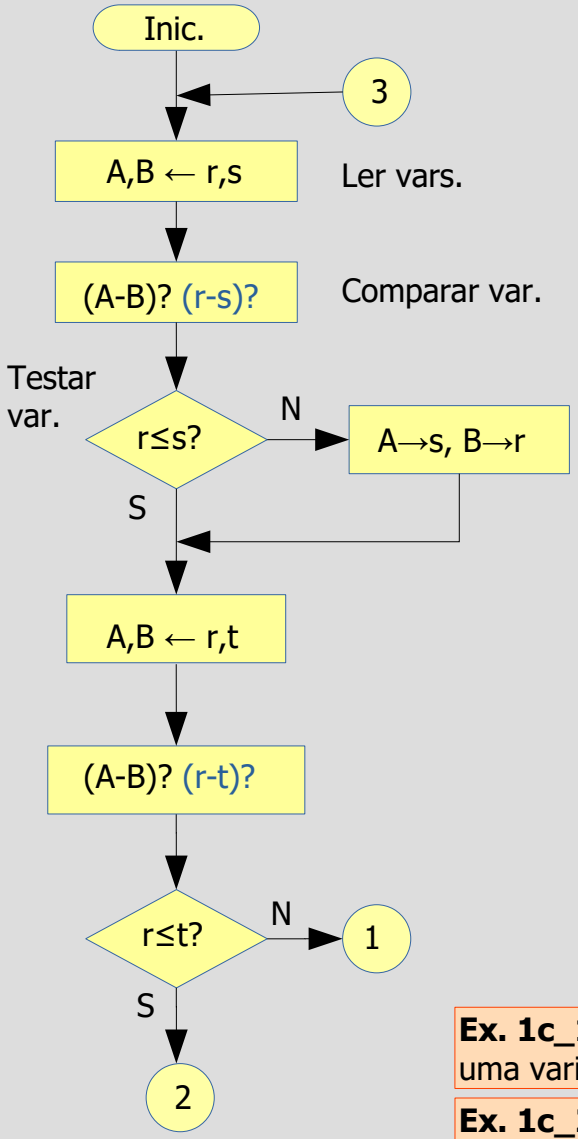
Ex. 1b_2 - Refaça (Ex.1b) considerando vr1 como sendo uma half-word. (Uso flag N).

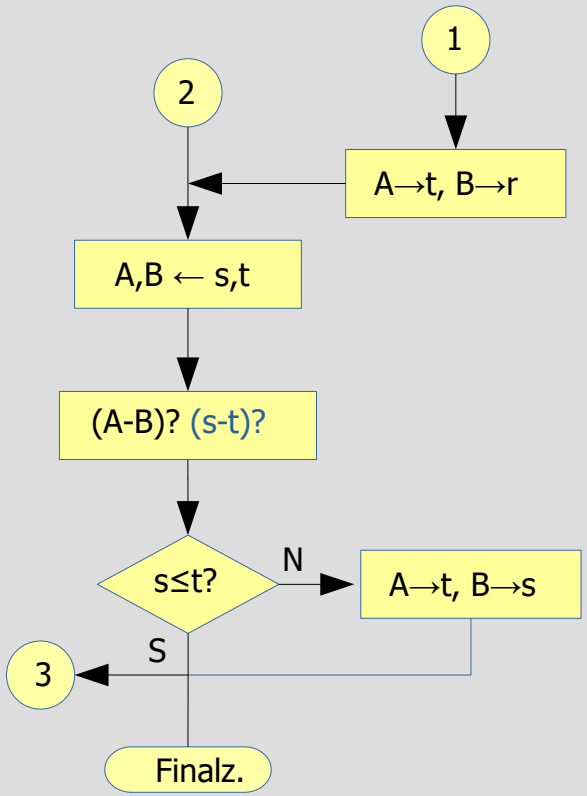
T4.6) BRANCH CONDICIONAL -> Exemplo

Ex. 1c - Seja g uma variável com 3 bytes, os quais são binários não sinalizados. Coloque em ordem crescente os bytes da variável g. O programa é cíclico. Faça as designações e alocações necessárias. Não usar variáveis extras. (Uso de instruções aritméticas; branch unsigned; flag C; A e B representam registros)



r≤s≤t, r≤t≤s, s≤r≤t, s≤t≤r, t≤r≤s, t≤s≤r





Ex. 1c_1 – Refaça o (Ex.1c) considerando que g é uma variável de 3 half-words. Teste-o.

Ex. 1c_2 – Refaça o (Ex.1c) considerando que g é uma variável de 3 words. Teste-o.