PCS3432 - Laboratório de Processadores

Relatório - E4

Bancada B8	
Bruno Mariz	11261826
Roberta Andrade	11260832

4.5.1 Assignments with operands in memory

Assume an array of 25 words. A compiler associates variables x and y with registers r0 and r1, respectively. Assume that the base address for the array is located in r2.

Translate this C statement/assignment using the post-indexed form:

> x = array[5] + y

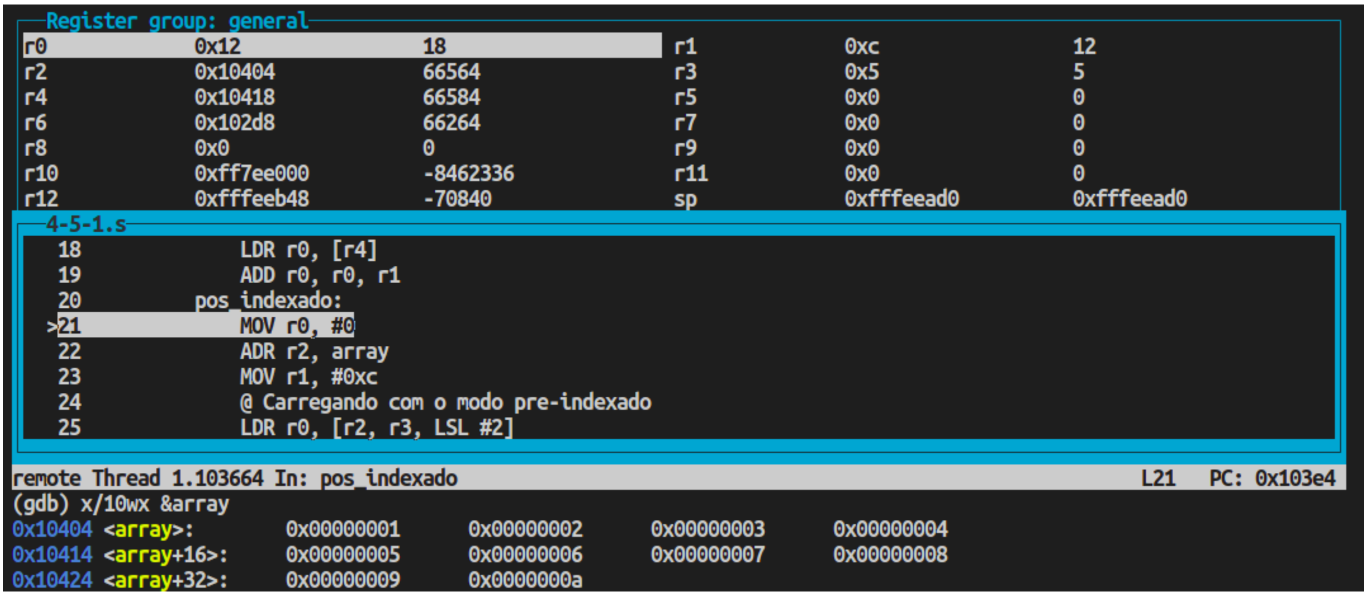
Now try writing it using the pre-indexed form

Código utilizado no exercício:

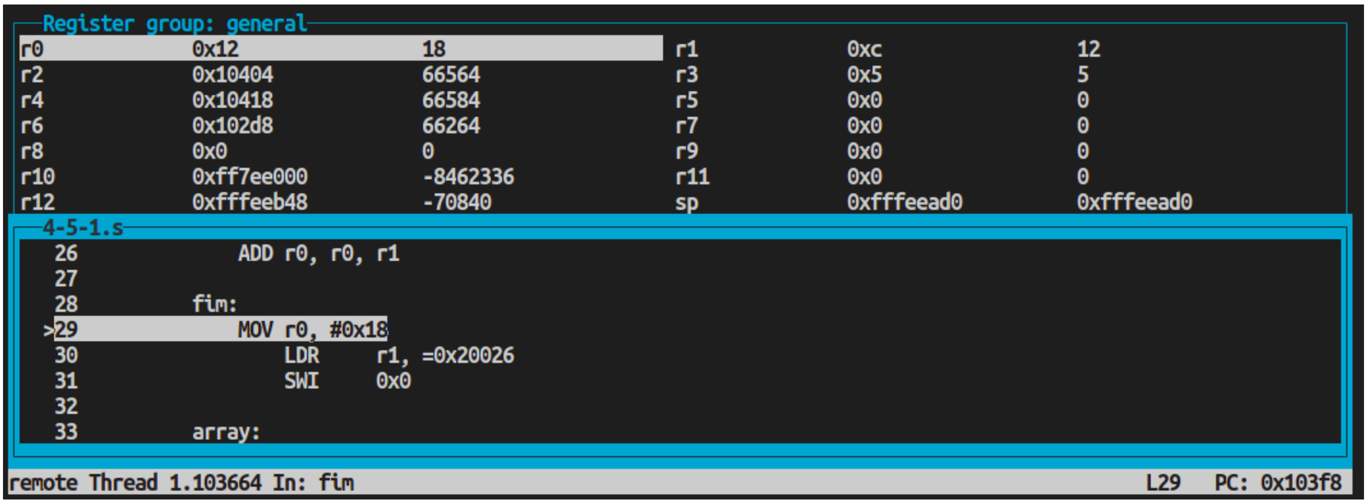
```
@ 4.5.1
.text
.global main
main:
    @ r0: x
    @ r1: y
    @ r2: array
    @ r3: indice
    @ r4: tmp
    @ x = array[5] + y
    MOV r3, #5
pos_indexado:
    MOV r0, #0
    ADR r2, array
    MOV r1, #0xc
    @ Carregando com o modo pos-indexado
    ADD r4, r2, r3, LSL #2
    LDR r0, [r4], #0
    ADD r0, r0, r1
pre_indexado:
    MOV r0, #0
    ADR r2, array
    MOV r1, #0xc
    @ Carregando com o modo pre-indexado
    LDR r0, [r2, r3, LSL #2]
```

```
    ADD r0, r0, r1

fim:
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x0

array:
    .word 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa
```

Ao executar o código acima, foi possível observar o valor esperado no registrador r0 (x = array[5]+y, ou r0 = 6+12 = 18) tanto utilizando o modo pós indexado quanto utilizando o modo pré indexado, como é possível observar nas imagens a seguir:



Resultado utilizando modo pós indexado



Resultado utilizando modo pre indexado

4.5.2 Loads and stores

Assume an array of 25 words. A compiler associates y with r1. Assume that the base address for the array is located in r2. Translate this C statement/assignment using the post-indexed form:

```
array[10] = array[5] + y
```

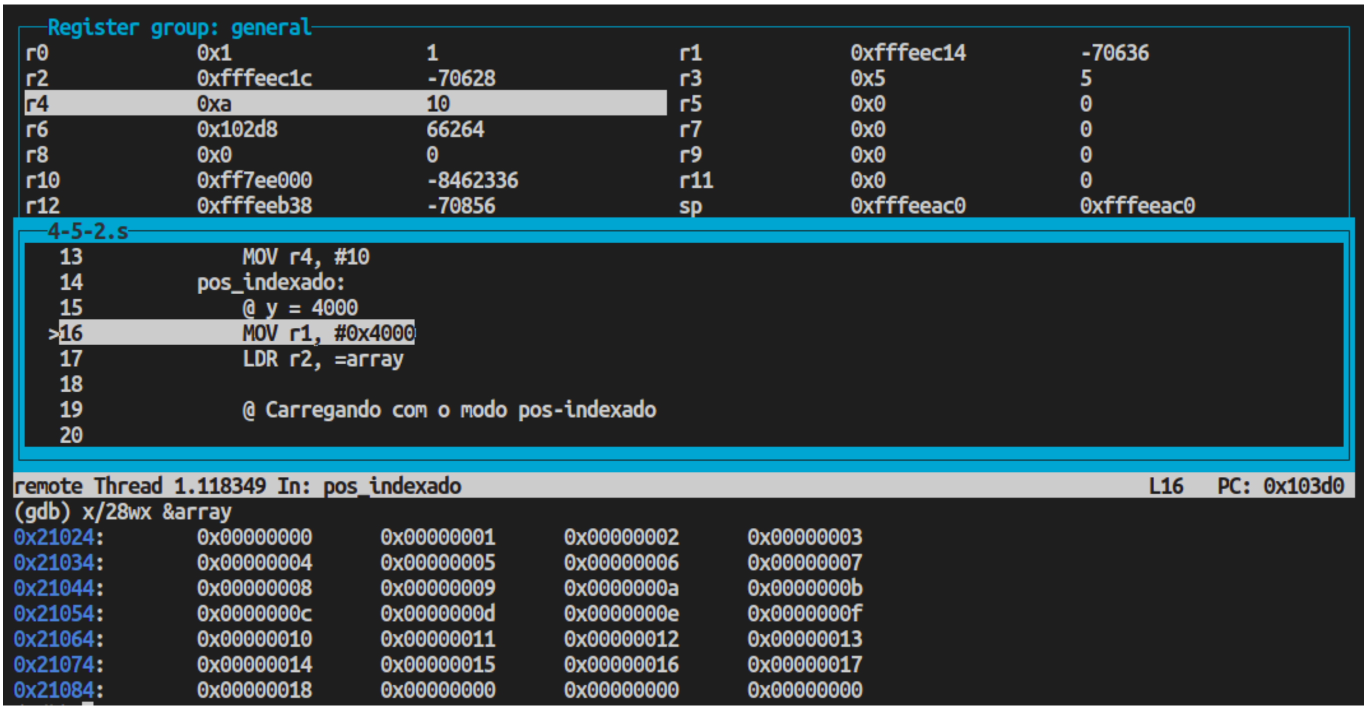
Now try it using the pre-indexed form.

Código utilizado no exercício:

```
@ 4.5.2
.text
.global main
main:
    @ r1: y
    @ r2: array
    @ r3: indice 1
    @ r4: indice 2
    @ r5: tmp
    @ r6: tmp
    @ array[10] = array[5] + y
    MOV r3, #5
    MOV r4, #10
pos_indexado:
    @ y = 4000
    MOV r1, #0x4000
    LDR r2, =array

    @ Carregando com o modo pos-indexado

    @ r5 = &(array[r3])
    ADD r5, r2, r3, LSL #2
    @ r5 = array[r3]
    LDR r5, [r5], #0
    @ r5 = array[r3] + y
    ADD r5, r5, r1
    @ r6 = &(array[r4])
    ADD r6, r2, r4, LSL #2
    @ array[r4] = array[r3] + y
    STR r5, [r6], #0

pre_indexado:
    @ Carregando com o modo pre-indexado

    @ Mudando indice 2 para variar do modo pos-indexado
    MOV r4, #11
    @ r5 = array[r3]
    LDR r5, [r2, r3, LSL #2]
    @ r5 = array[r3] + y
    ADD r5, r5, r1
```

```
    @ array[r4] = array[r3] + y
    STR r5, [r2, r4, LSL #2]

fim:
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x0

.data
array:
    .word 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xb, 0xc,
0xd, 0xe, 0xf, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18
```

Ao executar o código acima, foi possível observar o valor esperado no array após executar o comando de print `x/28wx &array`, tanto utilizando o modo pós indexado quanto utilizando o modo pré indexado, como é possível observar nas imagens a seguir:



Estado dos registradores e array antes de executar a instrução de store utilizando modo pós indexado

```
Register group: general
r0             0x1                 1                    r1             0x4000              16384
r2             0x21024             135204               r3             0x5                 5
r4             0xa                 10                   r5             0x4005              16389
r6             0x2104c             135244               r7             0x0                 0
r8             0x0                 0                    r9             0x0                 0
r10            0xff7ee000          -8462336             r11            0x0                 0
r12            0xfffeeb38          -70856               sp             0xfffeeac0          0xfffeeac0
4-5-2.s
    30              STR r5, [r6], #0
    31
    32          pre_indexado:
    33              @ Carregando com o modo pre-indexado
    34
    35              @ Mudando indice 2 para variar do modo pos-indexado
B+> 36              MOV r4, #11
    37              @ r5 = array[r3]
remote Thread 1.118349 In: pre_indexado                                                L36   PC: 0x103ec
(gdb) x/28wx &array
0x21024:        0x00000000      0x00000001      0x00000002      0x00000003
0x21034:        0x00000004      0x00000005      0x00000006      0x00000007
0x21044:        0x00000008      0x00000009      0x00004005      0x0000000b
0x21054:        0x0000000c      0x0000000d      0x0000000e      0x0000000f
0x21064:        0x00000010      0x00000011      0x00000012      0x00000013
0x21074:        0x00000014      0x00000015      0x00000016      0x00000017
0x21084:        0x00000018      0x00000000      0x00000000      0x00000000
(gdb)
```

Resultado utilizando modo pós indexado. É possível observar que a 11.a posição do array possui a soma de y com a 6.a posição do array.

Para a execução com o modo pré-indexado, foi alterada a posição onde foi guardado o resultado da soma para diferenciar do modo pós indexado, portanto o resultado foi salvo em `array[11]`:

```
Register group: general
r0             0x1                 1                    r1             0x4000              16384
r2             0x21024             135204               r3             0x5                 5
r4             0xb                 11                   r5             0x4005              16389
r6             0x2104c             135244               r7             0x0                 0
r8             0x0                 0                    r9             0x0                 0
r10            0xff7ee000          -8462336             r11            0x0                 0
r12            0xfffeeb38          -70856               sp             0xfffeeac0          0xfffeeac0
4-5-2.s
    42              STR r5, [r2, r4, LSL #2]
    43
    44          fim:
   >45              MOV r0, #0x18
    46                  LDR     r1, =0x20026
    47                  SWI     0x0
    48
    49          .data
remote Thread 1.118349 In: fim                                                         L45   PC: 0x103fc
(gdb) x/28wx &array
0x21024:        0x00000000      0x00000001      0x00000002      0x00000003
0x21034:        0x00000004      0x00000005      0x00000006      0x00000007
0x21044:        0x00000008      0x00000009      0x00004005      0x00004005
0x21054:        0x0000000c      0x0000000d      0x0000000e      0x0000000f
0x21064:        0x00000010      0x00000011      0x00000012      0x00000013
0x21074:        0x00000014      0x00000015      0x00000016      0x00000017
0x21084:        0x00000018      0x00000000      0x00000000      0x00000000
(gdb)
```

Resultado utilizando modo pré indexado. É possível observar que a 12.a posição do array possui a soma de y com a 6.a posição do array.

Para esse exercício, foi necessário separar o programa em área de texto e de dados para que fosse possível escrever o resultado no array, portanto foi necessário utilizar a instrução LDR em vez da instrução ADR.

4.5.3 Array assignment

Write ARM assembly to perform the following array assignment in C:

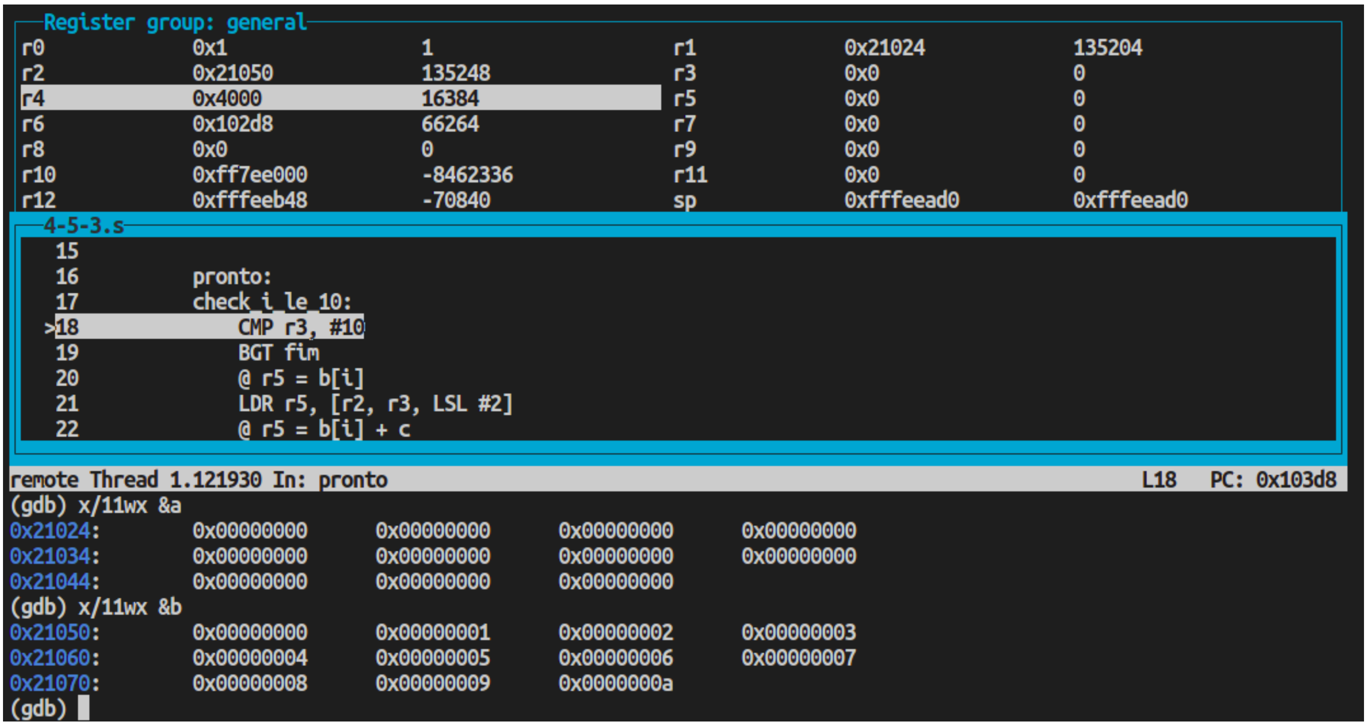
```
for ( i = 0; i <= 10; i++) {a[i] = b[i] + c;}
```

Assume that r3 contains i, r4 contains c, a starting address of the array a in r1, and a starting address of the array b in r2.

Código utilizado no exercício:

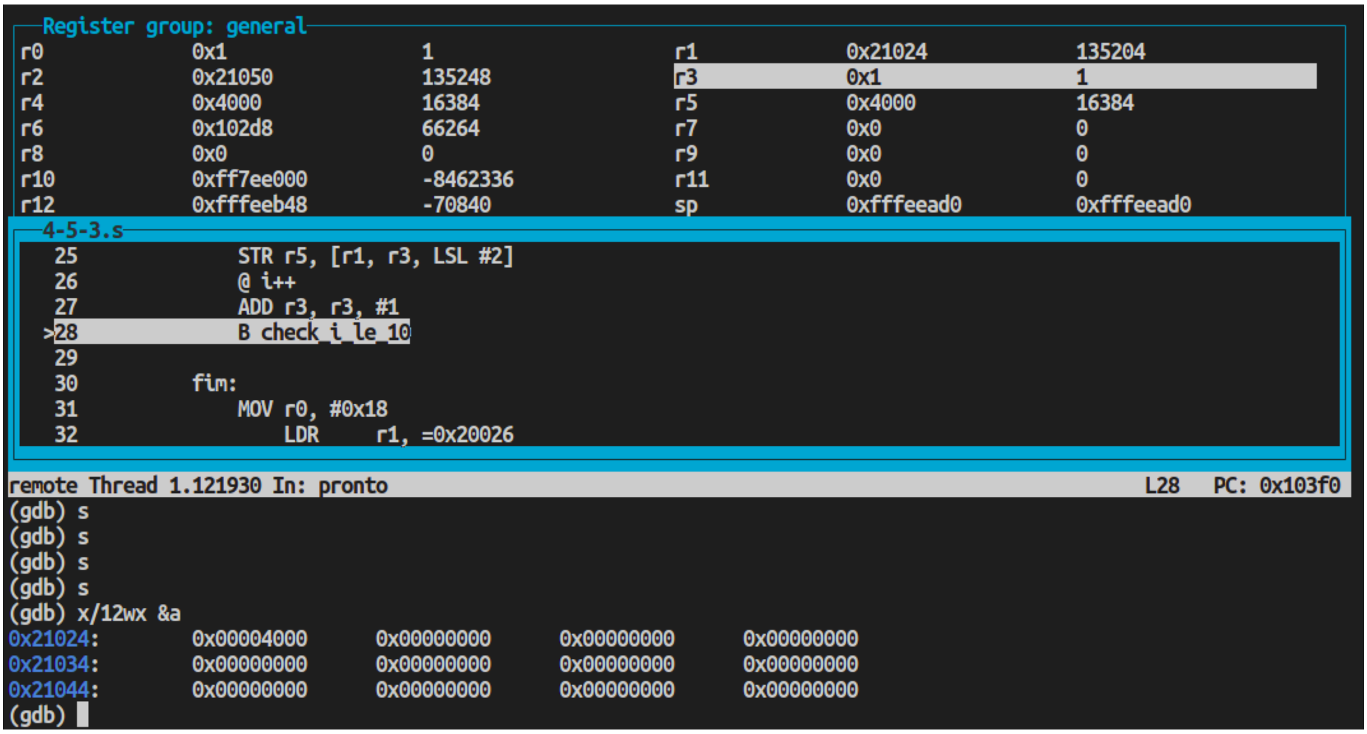
```
@ 4.5.3
.text
.global main
main:
    @ r1: a
    @ r2: b
    @ r3: i
    @ r4: c
    @ r5: tmp

    LDR r1, =a
    LDR r2, =b
    MOV r3, #0
    MOV r4, #0x4000

pronto:
check_i_le_10:
    CMP r3, #10
    BGT fim
    @ r5 = b[i]
    LDR r5, [r2, r3, LSL #2]
    @ r5 = b[i] + c
    ADD r5, r5, r4
    @ a[i] = r5
    STR r5, [r1, r3, LSL #2]
    @ i++
    ADD r3, r3, #1
    B check_i_le_10

fim:
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x0

.data
a:  .space 44
b:
    .word 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa
```

Estado dos registradores e arrays antes da execução:



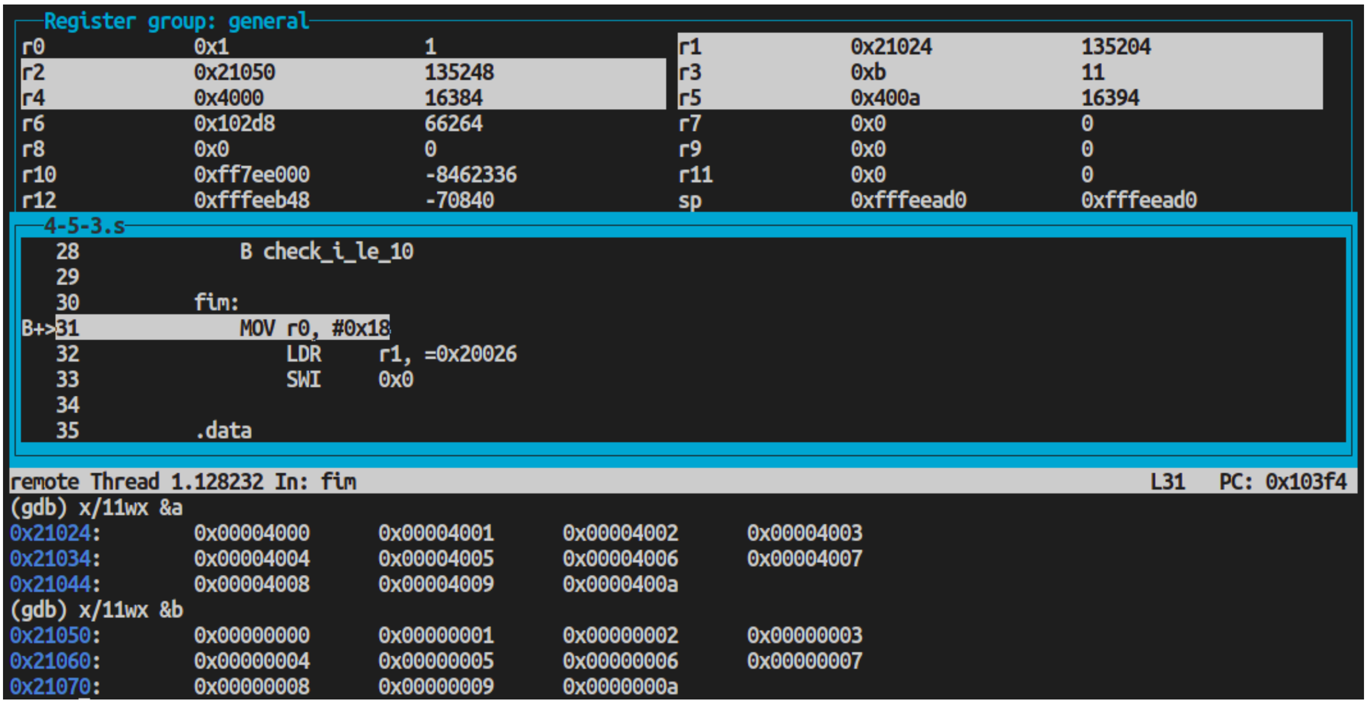
```
Register group: general
r0             0x1                 1                   r1             0x21024             135204
r2             0x21050             135248              r3             0x0                 0
r4             0x4000              16384               r5             0x0                 0
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
4-5-3.s
   15
   16          pronto:
   17          check_i_le_10:
  >18              CMP r3, #10
   19              BGT fim
   20              @ r5 = b[i]
   21              LDR r5, [r2, r3, LSL #2]
   22              @ r5 = b[i] + c
remote Thread 1.121930 In: pronto                                                       L18   PC: 0x103d8
(gdb) x/11wx &a
0x21024:       0x00000000      0x00000000      0x00000000      0x00000000
0x21034:       0x00000000      0x00000000      0x00000000      0x00000000
0x21044:       0x00000000      0x00000000      0x00000000
(gdb) x/11wx &b
0x21050:       0x00000000      0x00000001      0x00000002      0x00000003
0x21060:       0x00000004      0x00000005      0x00000006      0x00000007
0x21070:       0x00000008      0x00000009      0x0000000a
(gdb)
```

Estado após a primeira iteração:



```
Register group: general
r0             0x1                 1                   r1             0x21024             135204
r2             0x21050             135248              r3             0x1                 1
r4             0x4000              16384               r5             0x4000              16384
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
4-5-3.s
   25              STR r5, [r1, r3, LSL #2]
   26              @ i++
   27              ADD r3, r3, #1
  >28              B check_i_le_10
   29
   30          fim:
   31              MOV r0, #0x18
   32                  LDR     r1, =0x20026
remote Thread 1.121930 In: pronto                                                       L28   PC: 0x103f0
(gdb) s
(gdb) s
(gdb) s
(gdb) s
(gdb) x/12wx &a
0x21024:       0x00004000      0x00000000      0x00000000      0x00000000
0x21034:       0x00000000      0x00000000      0x00000000      0x00000000
0x21044:       0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

Estado ao final do loop:



4.5.4 Arrays and pointers

Consider the following two C procedures, which initialize an array to zero using a) indices, and b) pointers:

a)

```
init_Indices (int a[], int s) {
    int i;
    for ( i = 0; i < s; i ++)
        a[i] = 0; }
```

b)

```
init_Pointers (int *array, int s) {
    int *p;
    for (p = &array[0]; p < &array[s]; p++)
        *p = 0; }
```

Convert these two procedures to ARM assembly. Put the starting address of the array in r1, s in r2, and i and p in r3. Assume that s > 0 and that you have an array of bytes.

Código utilizado no exercício:

```
@ 4.5.4
.text
```

```
.global main
main:
    @ r1: a
    @ r2: s
    @ r3: i/p
    @ r4: zero
    @ r5: &array[s]

    LDR r1, =a
    MOV r2, #0x10
    MOV r4, #0

    BL init_indices
    BL init_pointers
    B fim

init_indices:
    @ i = 0
    MOV r3, #0
check_i_lt_s:
    CMP r3, r2
    @ retorna da funcao
    MOVGE pc, lr
    STRB r4, [r1, r3]
    @ i++
    ADD r3, r3, #1
    B check_i_lt_s

init_pointers:
    @ p = &array[0]
    LDR r3, =array
    @ r5 = &array[s]
    ADD r5, r3, r2
check_p_lt_array_s:
    CMP r3, r5
    MOVGE pc, lr
    STRB r4, [r3], #1
    B check_p_lt_array_s

fim:
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x0

.data
a:  .byte 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa

array: .byte 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a,
0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10, 0x11, 0x12, 0x13, 0x14
```

a)

Estado dos registradores e memória antes da execução do loop:

```
Register group: general
r0             0x1                 1                   r1             0x21024             135204
r2             0x10                16                  r3             0x0                 0
r4             0x0                 0                   r5             0x0                 0
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
4-5-4.s
   15              @ i = 0
   16              MOV r3, #0
   17          check_i_lt_s:
B+>18              CMP r3, r2
   19              BGE fim
   20              STRB r4, [r1, r3]
   21              @ i++
   22              ADD r3, r3, #1
remote Thread 1.135591 In: check_i_lt_s                                              L18   PC: 0x103d8
0x21024:        0xaaaaaaaa      0xaaaaaaaa      0xaaaaaaaa      0xaaaaaaaa
0x21034:        0xaaaaaaaa
```

Estado dos registradores e memória após a execução do loop:

```
Register group: general
r0             0x1                 1                   r1             0x21024             135204
r2             0x10                16                  r3             0x10                16
r4             0x0                 0                   r5             0x0                 0
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
4-5-4.s
   23              B check_i_lt_s
   24
   25          fim:
B+>26              MOV r0, #0x18
   27                  LDR     r1, =0x20026
   28                  SWI     0x0
   29
   30          .data
remote Thread 1.135591 In: fim                                                       L26   PC: 0x103ec
Breakpoint 3, fim () at 4-5-4.s:26
(gdb) x/5wx &a
0x21024:        0x00000000      0x00000000      0x00000000      0x00000000
0x21034:        0xaaaaaaaa
```

Note que o array a foi inicializado com 20 bytes 0xaa, porém o código utiliza s=16 para facilitar a visualização da parada do loop, portanto apenas os 16 primeiros bytes foram alterados para 0 pela subrotina.

b) Estado dos registradores e memória antes da execução do loop:

```
Register group: general
r0             0x1                 1                   r1             0x21024             135204
r2             0x10                16                  r3             0x21038             135224
r4             0x0                 0                   r5             0x21048             135240
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
4-5-4.s
   34              @ r5 = &array[s]
   35              ADD r5, r3, r2
   36          check_p_lt_array_s:
B+ 37              CMP r3, r5
  >38              MOVGE pc, lr
   39              STRB r4, [r3], #1
   40              B check_p_lt_array_s
   41
remote Thread 1.140790 In: check_p_lt_array_s                                   L38   PC: 0x10404
(gdb) x/5wx &array
0x21038:       0x04030201      0x08070605      0x0c0b0a09      0x100f0e0d
0x21048:       0x14131211
```

Estado dos registradores e memória após a execução do loop:

```
Register group: general
r0             0x1                 1                   r1             0x21024             135204
r2             0x10                16                  r3             0x21048             135240
r4             0x0                 0                   r5             0x21048             135240
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
4-5-4.s
   14
   15              BL init_indices
   16              BL init_pointers
  >17              B fim
   18
   19          init_indices:
   20              @ i = 0
   21              MOV r3, #0
remote Thread 1.140790 In: main                                                 L17   PC: 0x103dc
(gdb) x/5wx &array
0x21038:       0x00000000      0x00000000      0x00000000      0x00000000
0x21048:       0x14131211
```

Note que o array "array" foi inicializado com 20 bytes 0x01, 0x02, ..., 0x14, porém o código utiliza s=16 para facilitar a visualização da parada do loop, portanto apenas os 16 primeiros bytes foram alterados para 0 pela subrotina.

4.5.5 The Fibonacci sequence

The Fibonacci sequence is an infinite sequence of numbers such that:

```
f(0) = 0
f(1) = 1
f(2) = 1
f(3) = 2
f(4) = 3
f(5) = 5
f(6) = 8
.
```

```
.
.
f(n) = f(n - 1) + f(n - 2).
```

Write an ARM assembly program that computes the first 12 numbers of the sequence and stores the sequence in memory locations 0x4000 to 0x400B. Assume everything can be in bytes, because f(12) is the first number of the sequence that falls out of the byte range. You must use a loop, and only f(0) and f(1) can be stored outside the loop.

Código utilizado no exercício:

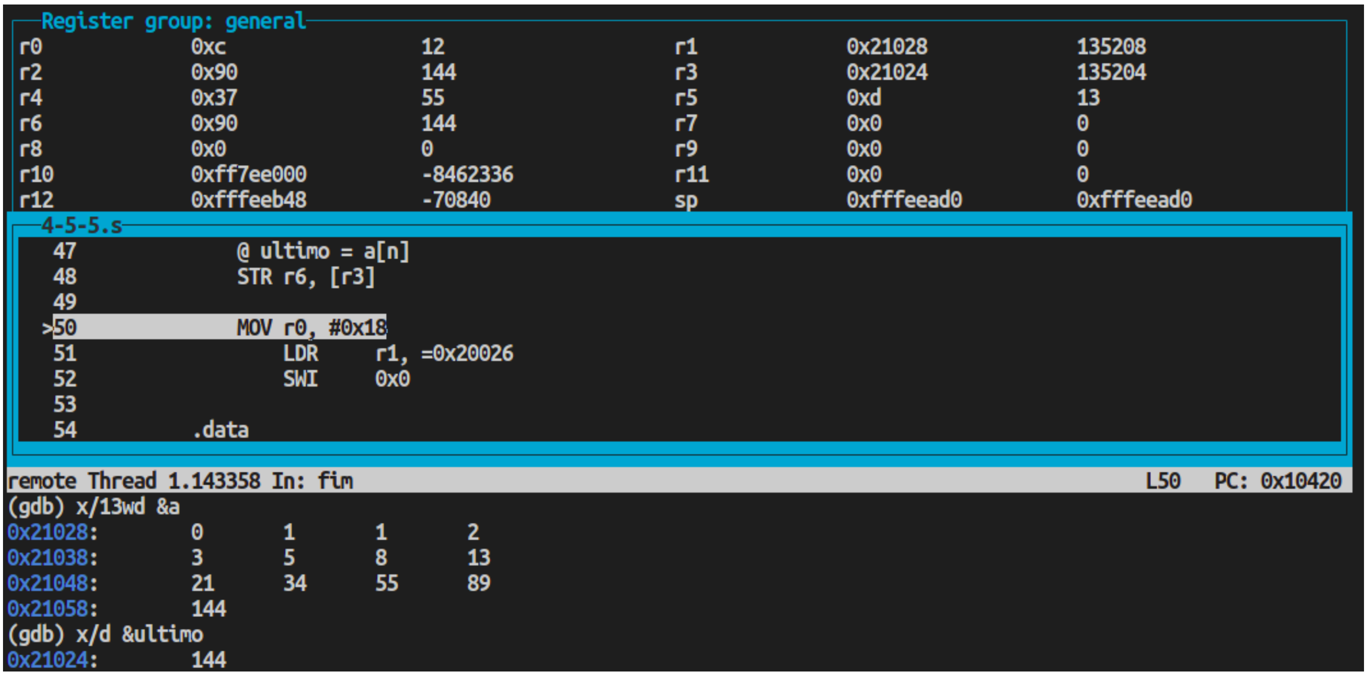
```
@ 4-5-5 Fibonacci
.text
.global main
main:
    @ r0: n
    @ r1: a
    @ r2: a[i]
    @ r3: a[i-1], tmp
    @ r4: a[i-2], tmp
    @ r5: indice
    @ r6: tmp
    MOV r0, #12
pronto:
    @ r1 = a
    LDR r1, =a
    @ i = 0
    MOV r5, #0
    @ Inicializar a[0] e a[1] com 0 e 1
    MOV r6, #0
    STR r6, [r1, r5, LSL #2] @ a[0] = 0
    ADD r5, r5, #1 @ i++
    MOV r6, #1
    STR r6, [r1, r5, LSL #2] @ a[1] = 1
    ADD r5, r5, #1 @ i++
check_i_gt_n:
    CMP r5, r0
    BGT fim
    @ r3 = a[i-1]
    SUB r3, r5, #1
    LDR r3, [r1, r3, LSL #2]
    @ r4 = a[i-2]
    SUB r4, r5, #2
    LDR r4, [r1, r4, LSL #2]
    @ r2 = a[i-1] + a[i-2]
    ADD r2, r3, r4
    @ a[i] = r2
    STR r2, [r1, r5, LSL #2]
    @ i++
    ADD r5, r5, #1
    B check_i_gt_n
```

```
fim:
    @ r6 = a[n]
    LDR r6, [r1, r0, LSL #2]
    @ r3 = &ultimo
    LDR r3, =ultimo
    @ ultimo = a[n]
    STR r6, [r3]

    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x0

.data
ultimo: .word 0
@ Array com sequencia de fibonacci
a:
    .space 400
```

É possível observar o estado dos registradores e do array no início do programa no print abaixo:

```
-Register group: general-
r0             0xc                 12                  r1             0xfffeec24          -70620
r2             0xfffeec2c          -70612              r3             0x103c8             66504
r4             0x10438             66616               r5             0x0                 0
r6             0x102d8             66264               r7             0x0                 0
r8             0x0                 0                   r9             0x0                 0
r10            0xff7ee000          -8462336            r11            0x0                 0
r12            0xfffeeb48          -70840              sp             0xfffeead0          0xfffeead0
-4-5-5.s-
B+ 12              MOV r0, #12
   13          pronto:
   14              @ r1 = a
  >15              LDR r1, =a
   16              @ i = 0
   17              MOV r5, #0
   18              @ Inicializar a[0] e a[1] com 0 e 1
   19              MOV r6, #0
remote Thread 1.143358 In: pronto                                                     L15   PC: 0x103cc
(gdb) x/13wx &a
0x21028:        0x00000000      0x00000000      0x00000000      0x00000000
0x21038:        0x00000000      0x00000000      0x00000000      0x00000000
0x21048:        0x00000000      0x00000000      0x00000000      0x00000000
0x21058:        0x00000000
```

Ao fim do programa, é possível observar a sequência de fibonacci nas posições de memória relativas ao array `a` utilizando o comando `x/13wd &a`, e o valor de f(n) com n=12 na posição de memória referente à label `ultimo`, com o comando `x/d &ultimo`:



4.5.6 The nth Fibonacci number

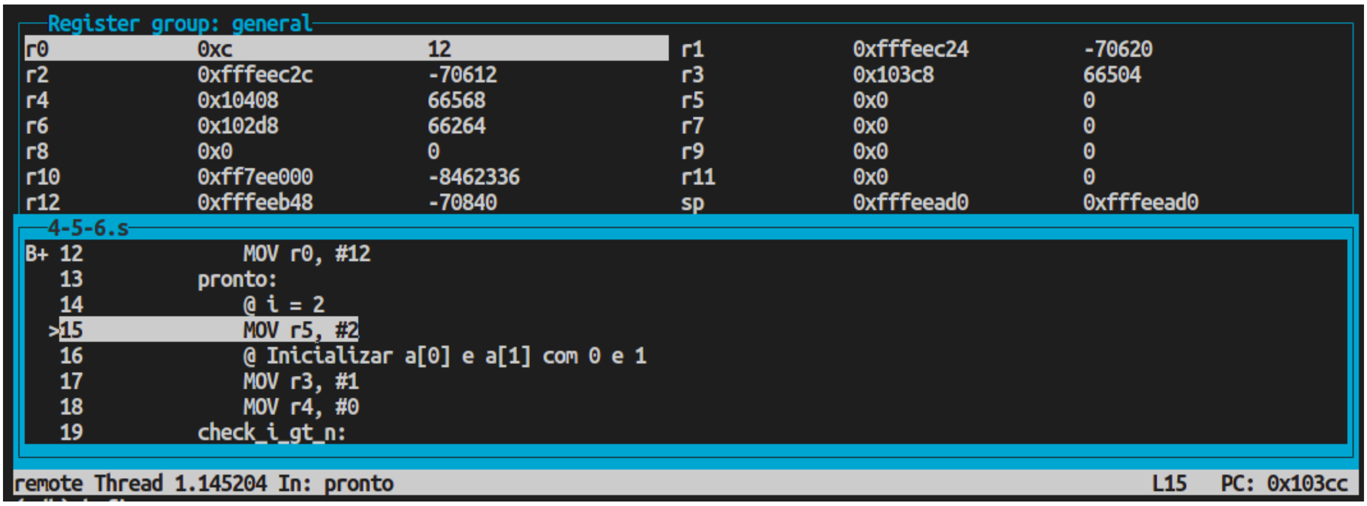
See The Fibonacci sequence and write ARM assembly to compute f(n). Start with r1 = n. At the end of the program, r0 = f(n).

Código utilizado no exercício:

```
@ 4-5-6 Nth Fibonacci
.text
.global main
main:
    @ r0: n, f(n)
    @ r1: a
    @ r2: a[i]
    @ r3: a[i-1], tmp
    @ r4: a[i-2], tmp
    @ r5: indice
    @ r6: tmp
    MOV r0, #12
pronto:
    @ i = 2
    MOV r5, #2
    @ Inicializar a[0] e a[1] com 0 e 1
    MOV r3, #1
    MOV r4, #0
check_i_gt_n:
    CMP r5, r0
    BGT fim
    @ r2 = a[i-1] + a[i-2]
    ADD r2, r3, r4
    @ Atualizar valores
    @ r4 = a[i-1]
    MOV r4, r3
```

```
    @ r3 = a[i]
    MOV r3, r2
    @ i++
    ADD r5, r5, #1
    B check_i_gt_n

fim:
    @ r0 = f(n)
    MOV r0, r2

    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x0

.data
ultimo: .word 0
@ Array com sequencia de fibonacci
a:
    .space 400
```

Estado dos registradores antes da execução do código:



Após a execução do loop, é possível observar o valor correspondente a f(12) = 144 em r0: