

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO



Experiência 4

Relatório

Turma: 02

25/04/2023

Nome	Número USP
Natanael Magalhães Cardoso	8914122
Tomas Gorescu Caldeira	9300881

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*

Forma Pós-indexada

A listagem abaixo mostra o programa implementado na forma pré-indexada

```
@ Descrição do algoritmo
@ -----
@ Item 4.5.1
@ Implementação da operação x = array[5] + y
@ na forma pós-indexada
@
@
@ Lista de Registradores
@ -----
@ r1 : valor y na operação acima
@ r2 : endereço base do array
@ r3 : posição do array a ser acessada
@ r0 : resultado final da operação
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_1_pos.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    LDR r2, =array           @ r2 = array
    MOV r1, #1               @ r1 = 1 (valor arbitrário para y)
```

```

MOV r3, #5                @ posição do array a ser acessada
LDR r0, [r2], r3, LSL #2  @ r0 = array[0], mas passa r0 para
apontar para array[5]
LDR r0, [r2]              @ r0 = array[5]
ADD r0, r0, r1            @ r0 = r0 + r1 = array[5] + y

fim:
SWI 0x123456             @ termina execução

array:
.word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19
.space 100 @ abrindo um espaço de 100 bytes para armazenar a array

```

Para a escrita do problema acima de maneira pós-indexada fizemos uma leitura da memória anterior à solicitada, para preparar o registrador r0 com o endereço de array[5]. Dessa maneira podemos treinar o funcionamento de acessos pós-indexados para “preparar” o próximo acesso a memória.

The screenshot shows a terminal window with the following content:

```

Activities  Terminal  abr 25 08:24  tomas@tomas-Latitude-3420: ~/LabDig/E4

Register group: general
r0 0x6 r1 0x1 1
r2 0x1041c 66588 r3 0x5 5
r4 0x3ffa0000 1073348608 r5 0x1 1
r6 0x2fff14 196372 r7 0x3ffff048 1073737800
r8 0x103ec 66540 r9 0x3fffd78 1073737080
r10 0x0 0 r11 0x2fff14 196372
r12 0x3ffa0000 1073348608 SP 0x407ffe80 0x407ffe80
lr 0x3fe2e3f8 1071834104 PC 0x10404 0x10404 <fim>
cpsr 0x60000010 1610612752 tpscr 0x0 0
fpsid 0x410430f0 1090793712 tpxc 0x40000000 1073741824

Item-4.5.1-pos.s
3
4 main: LDR r2, =array @ r2 = array
5 @ r1 = 1 (valor arbitrário para y)
6 MOV r1, #1
7 MOV r3, #5 @ posição do array a ser acessada
8 LDR r0, [r2], r3, LSL #2 @ r0 = array[0], mas passa r0 para apontar para array[5]
9 LDR r0, [r2] @ r0 = array[5]
10 ADD r0, r0, r1 @ r0 = r0 + r1 = array[5] + y
11 fim:
12 SWI 0x123456
13

remote Thread 1.60580 In: fim L12 PC: 0x10404
(gdb) c
Continuing.

Breakpoint 2, main () at item-4.5.1-pos.s:5
(gdb) s
(gdb) s
fin () at item-4.5.1-pos.s:12
(gdb) x/16 array
0x10408 <array>: 0 1 2 3
0x10418 <array+16>: 4 5 6 7
0x10428 <array+32>: 8 9 10 11
0x10438 <array+48>: 12 13 14 15
(gdb)

```

Forma pré-indexada

A listagem abaixo mostra o programa implementado na forma pré-indexada

```
@ Descrição do algoritmo
@ -----
@ Item 4.5.1
@ Implementação da operação x = array[5] + y
@ na forma pré-indexada
@
@
@ Lista de Registradores
@ -----
@ r1 : valor y na operação acima
@ r2 : endereço base do array
@ r3 : posição do array a ser acessada
@ r0 : resultado final da operação
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_1_pre.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    LDR r2, =array           @ r2 = array
    MOV r1, #1               @ r1 = 1 (valor arbitrário para y)
    MOV r3, #5               @ posição do array a ser acessada
    LDR r0, [r2, r3, LSL #2] @ r0 = array[5]
    ADD r0, r0, r1           @ r0 = r0 + r1 = array[5] + y

fim:
    SWI 0x123456             @ termina a execução
```

```
array:
.word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19
.space 100 @abrindo um espaço de 100 bytes para armazenar a array
```

Para a escrita do problema acima de maneira pré-indexada, podemos acessar a posição diretamente. No caso, salvamos a constante 5 em r3 e fizemos um LSL #2, que equivale a multiplicar por 4, para pularmos 5 words, passando de array[0] para array[5].

Me parece que em situações de acesso direto a uma posição da memória, a notação pré-fixada é mais direta.

The screenshot shows a terminal window with the following content:

```

Register group: general
r0      0x6      6      r1      0x1      1      r2      0x10404      66564
r3      0x5      5      r4      0x3ffa0000      1073348608      r5      0x1      1
r6      0x2fff14      196372      r7      0x3ffff048      1073737800      r8      0x103ec      66540
r9      0x3fffd78      1073737080      r10     0x0      0      r11     0x2ff14      196372
r12     0x3ffa0000      1073348608      sp      0x407ffe80      0x407ffe80      lr      0x3fe2e3f8      1071834104
pc      0x10400      0x10400 <fin>      cpsr     0x60000010      1610612752      fpscr     0x0      0
fpsid   0x410430f0      1090793712      fpexc   0x40000000      1073741824      AFSR0_EL1 0x0      0
AFSR1_EL1 0x0      0      DBGDIOR 0x3515f021      890630177      DBGDSAR   0x0      0
DBGBVR  0x0      0      DBGBCR   0x0      0      DBGWVR    0x0      0
DBGWCR   0x0      0      PAR      0x0      0      DBGWVR    0x0      0

item-4.5.1.s
2  .global main
3
4  main:
5      LDR r2, =array      @ r2 = array
6      MOV r1, #1          @ r1 = 1 (valor arbitrário para y)
7      MOV r3, #5          @ posição do array a ser acessada
8      LDR r0, [r2, r3, LSL #2] @ r0 = array[5]
9      ADD r0, r0, r1      @ r0 = r0 + r1 = array[5] + y
10 fin:
11      SWI 0x123456

remote Thread 1.59790 In: fin
(gdb) s
(gdb) x/16 arra
No symbol "arra" in current context.
(gdb) x/16 array
0x10404 <array>: 0 1 2 3
0x10414 <array+16>: 4 5 6 7
0x10424 <array+32>: 8 9 10 11
0x10434 <array+48>: 12 13 14 15
(gdb) s
(gdb) s
(gdb) s
(gdb) s
fin () at item-4.5.1.s:11
(gdb)

```

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:

$\text{array}[10] = \text{array}[5] + y$

Now try it using the pre-indexed form.

Forma pós-indexada

A listagem abaixo mostra a implementação do programa usando a forma pós-indexada

```
@ Descrição do algoritmo
@ -----
@ Implementação da operação array[10] = array[5] + y
@ na forma pós-indexada
@
@
@ Lista de Registradores
@ -----
@ r1 : valor y na operação acima
@ r2 : endereço base do array
@ r8 : valor do índice de acesso e armazenamento
@ r7 : registrador de trabalho
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_2_pos.s (montagem)
@ arm debug (depuração)

.text
.globl main

main:
    LDR r1, =100          @ valor de y
    LDR r2, =arr          @ posição base do array
```

```

LDR r8, =5 @ índice de load

ADD r2, r2, r8, LSL #2 @ calcula a posição de memória relativa
ao índice 5 (r8) do vetor

LDR r7, [r2], r8, LSL #2 @ carrega valor de arr[5] no reg. de
trabalho e desloca r2 para próximo índice

ADD r7, r7, r1 @ adiciona o valor de y em r7
STR r7, [r2], #0 @ armazena o valor de r7 em arr[10]
SWI 0x0 @ termina o programa

.data
arr: .word
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
.space 100 @abrindo um espaço de 100 bytes para armazenar a array

```

As figuras abaixo mostram os valores do array antes e após a execução do algoritmo implementado, respectivamente. É notado que o valor arr[10] na segunda figura apresenta o valor esperado.

The screenshot shows a GDB debugger window with the following content:

Register group: general

r0	0x1	1	r1	0x64	100
r2	0x30028	196648	r3	0x103ec	66540
r4	0x3ff90000	1073283072	r5	0x1	1
r6	0x2ff14	196372	r7	0x3ffff048	1073737800
r8	0x5	5	r9	0x3fffd78	1073737080
r10	0x0	0	r11	0x2ff14	196372
r12	0x3ff90000	1073283072	sp	0x407ffc0	0x407ffc0
lr	0x3fe1e3f8	1071768568	pc	0x103f8	0x103f8 <main+12>

ex_4_5_2_pos.s

```

25 LDR r2, =arr
26 LDR r8, =5
27
> 28 ADD r2, r2, r8, LSL #2
29 LDR r7, [r2], r8, LSL #2
30 ADD r7, r7, r1
31 STR r7, [r2], #0
32 SWI 0x0
33

```

remote Thread 1.90472 In: main L28 PC: 0x103f8

(gdb) s

(gdb) x/25 &arr

0x30028:	0	1	2	3
0x30038:	4	5	6	7
0x30048:	8	9	10	11
0x30058:	12	13	14	15
0x30068:	16	17	18	19
0x30078:	20	21	22	23
0x30088:	24			

(gdb)

```
natan@asus: ~/repos/poli/lab-processadores/exp04

Register group: general
r0      0x1          1          r1      0x64          100
r2      0x30050      196688     r3      0x103ec      66540
r4      0x3ff90000   1073283072 r5      0x1          1
r6      0x2ff14      196372     r7      0x69          105
r8      0x5           5          r9      0x3fffed78   1073737080
r10     0x0           0          r11     0x2ff14      196372
r12     0x3ff90000   1073283072 sp      0x407ffc0       0x407ffc0
lr      0x3fe1e3f8   1071768568 pc      0x10408      0x10408 <main+28>

ex_4_5_2_pos.s
29     LDR r7, [r2], r8, LSL #2
30     ADD r7, r7, r1
31     STR r7, [r2], #0
> 32     SWI 0x0
33
34
35     .data
36     arr: .word 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25
37

remote Thread 1.90472 In: main L32 PC: 0x10408
(gdb) s
(gdb) s
(gdb) s
(gdb) s
(gdb) x/16 &arr
0x30028: 0 1 2 3
0x30038: 4 5 6 7
0x30048: 8 9 105 11
0x30058: 12 13 14 15
(gdb) 
```

Forma pré-indexada

A listagem abaixo mostra a implementação do algoritmo na forma pré-indexada.

```
@ Descrição do algoritmo
@ -----
@ Implementação da operação array[10] = array[5] + y
@ na forma pré-indexada
@
@
@ Lista de Registradores
@ -----
@ r1 : valor y na operação acima
@ r2 : endereço base do array
@ r8 : valor do índice de acesso
@ r9 : valor do índice de armazenamento
@ r7 : registrador de trabalho
@
@
@ Instruções de uso
```



```

@ -----
@ arm build ex_4_5_2_pre.s (montagem)
@ arm debug (depuração)

.text
.globl main

main:
    LDR r1, =100           @ valor de y
    LDR r2, =arr           @ posição base do array
    LDR r8, =5             @ índice de load
    LDR r9, =10            @ índice de store

    LDR r7, [r2, r8, LSL #2] @ calcula a posição de memória relativa
                             ao índice 5 (r8) do vetor e carrega arr[5] em r7
    ADD r7, r7, r1          @ adiciona o valor de y em r7
    STR r7, [r2, r9, LSL #2] @ calcula a posição na memória de
                             arr[10] e armazena o conteúdo de r7 em arr[10]
    SWI 0x0                @ termina o programa

.data
    arr: .word
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
    .space 100 @abrindo um espaço de 100 bytes para armazenar a array

```

As figuras abaixo mostram os valores do vetor antes e depois da execução do programa, respectivamente. Na segunda figura, o valor de arr[10] é mostrado como o esperado.

Na implementação na forma pós-indexada, foi usada uma instrução a mais, para calcular o endereço na memória relativo ao quinto elemento do array. No entanto, foi usado um registrador a menos, pois o valor do índice de escrita já é calculado no carregamento.

Assim, neste exemplo, foi notado que uma forma não substitui a outra, mas complementa, pois cada uma implica em características diferentes em relação ao uso de recursos computacionais e a escolha entre as formas é dependente da situação e dos recursos a serem otimizados.

```
natan@asus: ~/repos/poli/lab-processadores/exp04

Register group: general
r0      0x1      1      r1      0x64      100
r2      0x30028  196648  r3      0x103ec  66540
r4      0x3ff90000 1073283072 r5      0x1      1
r6      0x2fff14 196372  r7      0x3ffff048 1073737800
r8      0x5      5      r9      0xa      10
r10     0x0      0      r11     0x2fff14 196372
r12     0x3ff90000 1073283072 sp      0x407ffc0 0x407ffc0
lr      0x3fe1e3f8 1071768568 pc      0x103fc 0x103fc <main+16>

ex_4_5_2_pre.s
27 LDR r8, =5
28 LDR r9, =10
29
> 30 LDR r7, [r2, r8, LSL #2]
31 ADD r7, r7, r1
32 STR r7, [r2, r9, LSL #2]
33 SWI 0x0
34
35

remote Thread 1.90733 In: main L30 PC: 0x103fc
(gdb) s
(gdb) x/25 &arr
0x30028: 0 1 2 3
0x30038: 4 5 6 7
0x30048: 8 9 10 11
0x30058: 12 13 14 15
0x30068: 16 17 18 19
0x30078: 20 21 22 23
0x30088: 24
(gdb) 
```

```
natan@asus: ~/repos/poli/lab-processadores/exp04

Register group: general
r0      0x1      1      r1      0x64      100
r2      0x30028  196648  r3      0x103ec  66540
r4      0x3ff90000 1073283072 r5      0x1      1
r6      0x2fff14 196372  r7      0x69      105
r8      0x5      5      r9      0xa      10
r10     0x0      0      r11     0x2fff14 196372
r12     0x3ff90000 1073283072 sp      0x407ffc0 0x407ffc0
lr      0x3fe1e3f8 1071768568 pc      0x10408 0x10408 <main+28>

ex_4_5_2_pre.s
27 LDR r8, =5
28 LDR r9, =10
29
30 LDR r7, [r2, r8, LSL #2]
31 ADD r7, r7, r1
32 STR r7, [r2, r9, LSL #2]
> 33 SWI 0x0
34
35

remote Thread 1.90808 In: main L33 PC: 0x10408
(gdb) s
(gdb) x/25 &arr
0x30028: 0 1 2 3
0x30038: 4 5 6 7
0x30048: 8 9 105 11
0x30058: 12 13 14 15
0x30068: 16 17 18 19
0x30078: 20 21 22 23
0x30088: 24
(gdb) 
```

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.

A listagem abaixo mostra o programa implementado

```
.text
.global main

main:
    LDR    r1, =a                @ r1 = a
    LDR    r2, =b                @ r2 = b
    MOV    r3, #0                @ i = 0
    MOV    r4, #5                @ c = 5 (constante arbitraria)
    MOV    r6, r1                @ copia endereço de a

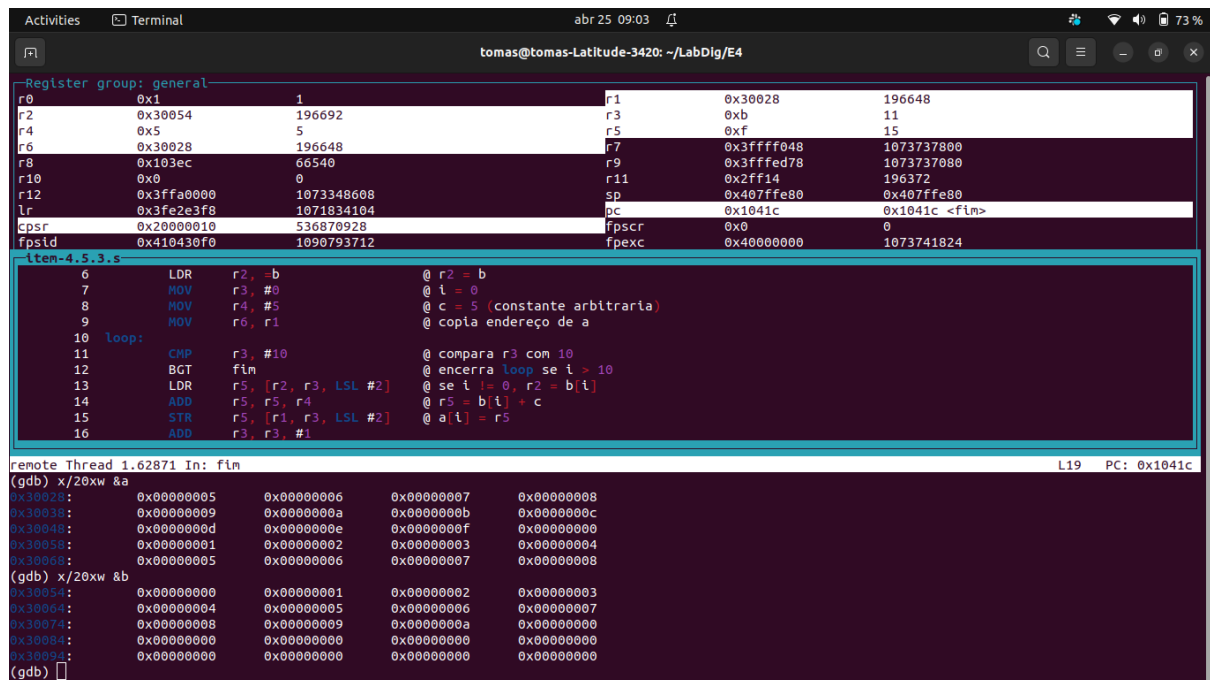
loop:
    CMP    r3, #10               @ compara r3 com 10
    BGT    fim                   @ encerra loop se i > 10
    LDR    r5, [r2, r3, LSL #2]  @ se i != 0, r2 = b[i]
    ADD    r5, r5, r4             @ r5 = b[i] + c
    STR    r5, [r1, r3, LSL #2]  @ a[i] = r5
    ADD    r3, r3, #1            @ incrementa r3 em 1 unidade
    B      loop                  @ retorna ao loop

fim:
    SWI    0x123456              @ termina a execução

.data
a:
    .word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
b:
    .word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

```
.space 100 @ abrindo um espaço de 100 bytes para armazenar a array
.align 1 @ align to even bytes REQUIRED!!!
```

A figura abaixo mostra a tela de depuração com os valores dos vetores a e b.



The screenshot shows a GDB terminal window with the following content:

```
Register group: general
r0      0x1      1      r1      0x30028      196648
r2      0x30054      196692      r3      0xb      11
r4      0x5      5      r5      0xf      15
r6      0x30028      196648      r7      0x3ffff048      1073737800
r8      0x103ec      66540      r9      0x3fffed78      1073737080
r10     0x0      0      r11     0x2ff14      196372
r12     0x3ffa0000      1073348608      sp      0x407ffe80      0x407ffe80
lr      0x3fe2e3f8      1071834104      pc      0x1041c      0x1041c <fln>
cpsr    0x20000010      536870928      fpscr   0x0      0
tpslid  0x410430f0      1090793712      tpexc   0x40000000      1073741824

Item-4.5.3.s
6      LDR    r2, =b      @ r2 = b
7      MOV    r3, #0      @ i = 0
8      MOV    r4, #5      @ c = 5 (constante arbitrária)
9      MOV    r6, r1      @ copia endereço de a
10     loop:
11     CMP    r3, #10      @ compara r3 com 10
12     BGT    fln          @ encerra loop se i > 10
13     LDR    r5, [r2, LSL #2] @ se i != 0, r2 = b[i]
14     ADD    r5, r5, r4      @ r5 = b[i] + c
15     STR    r5, [r1, LSL #2] @ a[i] = r5
16     ADD    r3, r3, #1

remote Thread 1.62871 In: fln
(gdb) x/20xw &a
0x30028: 0x00000005 0x00000006 0x00000007 0x00000008
0x30038: 0x00000009 0x0000000a 0x0000000b 0x0000000c
0x30048: 0x0000000d 0x0000000e 0x0000000f 0x00000000
0x30058: 0x00000001 0x00000002 0x00000003 0x00000004
0x30068: 0x00000005 0x00000006 0x00000007 0x00000008
(gdb) x/20xw &b
0x30054: 0x00000000 0x00000001 0x00000002 0x00000003
0x30064: 0x00000004 0x00000005 0x00000006 0x00000007
0x30074: 0x00000008 0x00000009 0x0000000a 0x00000000
0x30084: 0x00000000 0x00000000 0x00000000 0x00000000
0x30094: 0x00000000 0x00000000 0x00000000 0x00000000
(gdb)
```

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; }
```

```
@ Descrição do algoritmo
@ -----
@ Item 4.5.4.a:
@ Inicialia array com zeros usando índices
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_4_array.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    LDR    r1, =a                @ r1 = a
    MOV    r2, #5                @ s = 5
    MOV    r3, #0                @ i = 0
    MOV    r4, #0                @ constante 0

loop:
    CMP    r3, r2                @ compara r3 com r2
    BEQ    fim                  @ encerra loop se i = s
    STR    r4, [r1, r3, LSL #2]  @ a[i] = 0
    ADD    r3, r3, #1            @ incrementa 1
    B      loop                  @ retorna para o loop
```

```

fim:
    SWI    0x123456                @ termina o programa
    .data
a:
    .word 1, 2, 3, 4, 5

    .space 100 @abrindo um espaco de 100 bytes para armazenar a array
    .align 1 @ align to even bytes REQUIRED!!!

```

Para o caso do array, o loop é mais próximo do que parece “natural”. A condição de parada do loop é a comparação entre o índice i , armazenado em r ; e a constante s , armazenada em r .

A cada iteração, salvamos a constante 0 em a , usando o índice multiplicado por 4 para chegar ao endereço correto na iteração.

```

Register group: general
r0      0x1      1      r1      0x30028      196648
r2      0x5      5      r3      0x5      5
r4      0x0      0      r5      0x1      1
r6      0x2ff14      196372      r7      0x3ffff048      1073737800
r8      0x103ec      66540      r9      0x3fffed78      1073737080
r10     0x0      0      r11     0x2ff14      196372
r12     0x3ffa0000      1073348608      sp      0x407ffe80      0x407ffe80
lr      0x3fe2e3f8      1071834104      pc      0x10410      0x10410 <fim>
cpsr    0x60000010      1610612752      fpscr   0x0      0
fpsidr  0x410430f0      1090793712      fpexc   0x40000000      1073741824

item-4.5.4.s
12      STR    r4, [r1, r3, LSL #2] @ a[i] = 0
13      ADD    r3, r3, #1
14      B      loop
15  fim:
16      SWI    0x123456
17
18  .data
19  a:
20      .word 1, 2, 3, 4, 5
21
22  .space 100 @abrindo um espaco de 100 bytes para armazenar a array

remote Thread 1.65303 In: fim
(gdb) x/dw &a
0x30028: 1
(gdb) x/5dw &a
0x30028: 1 2 3 4
0x30030: 5
(gdb) c
Continuing.

Breakpoint 2, fim () at item-4.5.4.s:16
(gdb) x/5dw &a
0x30028: 0 0 0 0
0x30030: 0
(gdb)

```

```

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

```

```

@ Descrição do algoritmo
@ -----
@ Item 4.5.4.b:
@ Inicialia array com zeros usando ponteiros
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_4_pointer.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    LDR    r1, =a                @ r1 = a
    MOV    r2, #5                @ s = 5
    MOV    r4, #0                @ constante 0
    ADD    r2, r1, r2, LSL #2    @ r2 = &array[s]

loop:
    CMP    r1, r2                @ compara r1 com r2
    BEQ    fim                  @ encerra loop se i = s
    STR    r4, [r1]              @ a[i] = 0
    ADD    r1, r1, #4            @ incremente r1 com 1 word
    B      loop

                                @ retorna para o loop

fim:
    SWI    0x123456              @ termina o programa

.data
a:

```

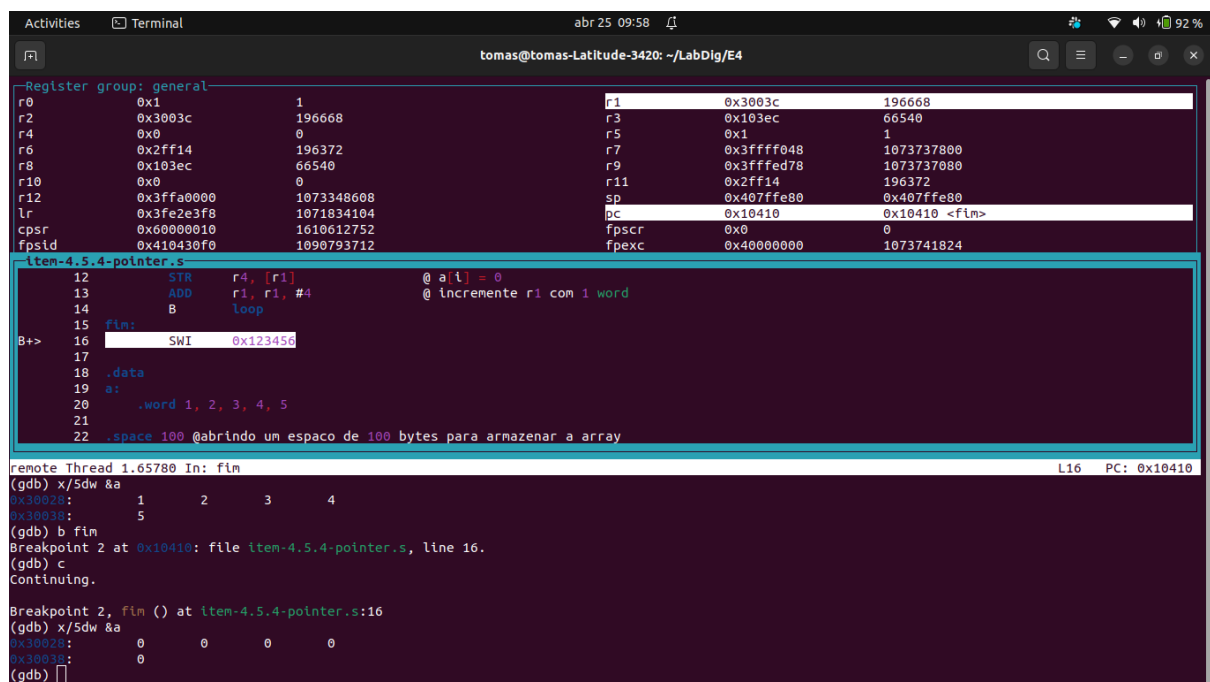
```
.word 1, 2, 3, 4, 5
```

```
.space 100 @abrindo um espaço de 100 bytes para armazenar a array  
.align 1 @ align to even bytes REQUIRED!!!
```

Já o caso do ponteiro, parece mais intuitivo no assembly do que no C, já que os endereços já são variáveis declaradas. Ao invés de compararmos índices e a constante, precisamos comparar 2 endereços de memória.

Para isso, precisamos calcular o endereço de `array[s]`. Para isso, somamos $4*s$ ao endereço inicial do array.

No loop, ao invés de somar 1 a um índice, somamos 4 ao endereço inicial de array.



The screenshot shows a terminal window with the title "tomas@tomas-Latitude-3420: ~/LabDig/E4". The terminal displays assembly code for a program named "item-4.5.4-pointer.s". The code includes a register group, a loop, and a function "fim". The GDB output shows the program running and then hitting a breakpoint at line 16 of "item-4.5.4-pointer.s". The GDB prompt shows the current register values and the program state.

```
Register group: general  
r0 0x1 1 r1 0x3003c 196668  
r2 0x3003c 196668 r3 0x103ec 66540  
r4 0x0 0 r5 0x1 1  
r6 0x2ff14 196372 r7 0x3ffff048 1073737800  
r8 0x103ec 66540 r9 0x3fffed78 1073737080  
r10 0x0 0 r11 0x2ff14 196372  
r12 0x3ffa0000 1073348608 sp 0x407ffe80 0x407ffe80  
lr 0x3fe2e3f8 1071834104 pc 0x10410 0x10410 <fim>  
cpsr 0x60000010 1610612752 fpscr 0x0 0  
fpsid 0x410430f0 1090793712 fpexc 0x40000000 1073741824  
  
item-4.5.4-pointer.s  
12 STR r4, [r1] @ a[l] = 0  
13 ADD r1, r1, #4 @ incremente r1 com 1 word  
14 B loop  
15 fim:  
16 SWI 0x123456  
17  
18 .data  
19 a:  
20 .word 1, 2, 3, 4, 5  
21  
22 .space 100 @abrindo um espaço de 100 bytes para armazenar a array  
  
remote Thread 1.65780 In: fim L16 PC: 0x10410  
(gdb) x/5dw &a  
0x30028: 1 2 3 4  
0x30038: 5  
(gdb) b fim  
Breakpoint 2 at 0x10410: file item-4.5.4-pointer.s, line 16.  
(gdb) c  
Continuing.  
Breakpoint 2, fim () at item-4.5.4-pointer.s:16  
(gdb) x/5dw &a  
0x30028: 0 0 0 0  
0x30038: 0  
(gdb) □
```


4.5.5 - The Fibonacci sequence

Nesse item, aproveitamos o problema para nos aprofundarmos nos estudos dos loads e dos stores.

O algoritmo começa inicializando o array de resultados do fibonacci. O único parâmetro recebido deve ser o número de iterações a serem realizadas. Fora do loop, salvamos $F(0)$ e $F(1)$ na memória. A cada iteração, carregamos da memória $f(n-2)$ e $f(n-1)$, calculamos $f(n)$ e o salvamos na memória.

```
@ Descrição do algoritmo
@ -----
@ Item 4.5.5
@ Implementação da sequência de Fibonacci armazenando
@ os valores calculados na memória
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_5.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    ldr    r0, =fibonaci @ carrega array em r0
    mov    r5, r0        @ só para consultar na memória depois
    mov    r1, #10       @ tamanho da sequência
    mov    r2, #0        @ f(0)
    mov    r3, #1        @ f(1)
    str    r2, [r0]      @ fib[0] = f(0)
    str    r3, [r0, #4]  @ fib[1] = f(1)
    sub    r1, r1, #1

loop:
    ldr    r2, [r0]      @ carrega primeiro da soma em r2
    add    r0, r0, #4    @ incrementa "ponteiro"
    ldr    r3, [r0]      @ carrega segundo da soma em r3
```

```

add    r4, r3, r2    @ soma os dois
str    r4, [r0, #4]  @ guarda r4 em r0
    subs    r1, r1, #1    @ decrementa contador
bne    loop          @ se contador =1, para, se não continua
b      fim
fim:
swi    0x0           @resultado final em r4

.data
fibonaci:
.word 0, 0, 0, 0, 0, 0, 0, 0
.space 100 @abrindo um espaço de 100 bytes para armazenar a array
.align 1 @ align to even bytes REQUIRED!!!

```

O print abaixo demonstra o funcionamento do algoritmo para o cálculo de $F(10)$:

```

Activities  Terminal  abr 25 09:31  81%
tomas@tomas-Latitude-3420: ~/LabDig/E4

Register group: general
r0 0x3004c 196684 r1 0x0 0 r2 0x15 21
r3 0x22 34 r4 0x37 55 r5 0x30028 196648
r6 0x2ff14 196372 r7 0x3ffff048 1073737800 r8 0x103ec 66540
r9 0x3fffd78 1073737080 r10 0x0 0 r11 0x2ff14 196372
r12 0x3ffa0000 1073348608 sp 0x407ffe80 0x407ffe80 lr 0x3fe2e3f8 1071834104
pc 0x1042c 0x1042c <fim> cpsr 0x60000010 1610612752 fpscr 0x0 0
fpsid 0x410430f0 1090793712 fpexc 0x40000000 1073741824 AFSR0_EL1 0x0 0
AFSR1_EL1 0x0 0 DBGDIOR 0x3515f021 890630177 DBGDSAR 0x0 0
DBGBVR 0x0 0 DBGBCR 0x0 0 DBGWVR 0x0 0
DBGWCR 0x0 0 PAR 0x0 0 DBGWVR 0x0 0

item-4.5.s
13 add r0, r0, #4 @ incrementa "ponteiro"
14 ldr r3, [r0] @ carrega segundo da soma em r3
15 add r4, r3, r2 @ soma os dois
16 str r4, [r0, #4] @ guarda r4 em r0
17
18 subs r1, r1, #1 @ decrementa contador
19 bne loop @ se contador =1, para, se não continua
20 b fim
21
22 fim:
23 swi 0x0 @resultado final em r4

B+>
remote Thread 1.63745 In: fim L23 PC: 0x1042c

Breakpoint 1, fim () at item-4.5.s:23
(gdb) x/16xw &fibonaci
0x30028: 0x00000000 0x00000001 0x00000001 0x00000002
0x30038: 0x00000003 0x00000005 0x00000008 0x0000000d
0x30048: 0x00000015 0x00000022 0x00000037 0x00000000
0x30058: 0x00000000 0x00000000 0x00000000 0x00000000
(gdb) x/16dw &fibonaci
0x30028: 0 1 1 2
0x30038: 3 5 8 13
0x30048: 21 34 55 0
0x30058: 0 0 0 0
(gdb)

```

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, $r0 = f(n)$.*

O algoritmo é quase igual ao anterior, mas escreve o resultado final em R0 ao final da execução.

```
@ Descrição do algoritmo
@ -----
@ Item 4.5.6
@ Implementação da sequência de Fibonacci. Com o resultado
@ final f(n) armazenado no registrador R0.
@
@
@ Instruções de uso
@ -----
@ arm build ex_4_5_6.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    ldr    r0, =fibonaci    @ carrega array em r0
    mov    r5, r0           @ só para consultar na memória depois
    mov    r1, #10          @ tamanho da sequência
    mov    r2, #0           @ f(0)
    mov    r3, #1           @ f(1)
    str    r2, [r0]         @ fib[0] = f(0)
    str    r3, [r0, #4]     @ fib[1] = f(1)
    sub    r1, r1, #1

loop:
    ldr    r2, [r0]         @ carrega primeiro da soma em r2
    add    r0, r0, #4       @ incrementa "ponteiro"
    ldr    r3, [r0]         @ carrega segundo da soma em r3
```

```

add    r4, r3, r2        @ soma os dois
str     r4, [r0, #4]      @ guarda r4 em r0
subs   r1, r1, #1        @ decrementa contador
bne     loop              @ se contador !=1, para, se não continua
b       fim
fim:
mov     r0, r4            @ resultado final em r0
swi     0x0               @ resultado final em r4

.data
fibonaci:
.word 0, 0, 0, 0, 0, 0, 0, 0
.space 100 @ abrindo um espaço de 100 bytes para armazenar a array
.align 1 @ align to even bytes REQUIRED!!!

```

No exemplo utilizado abaixo, para $F(10)$, podemos ver o resultado final (55), armazenado no registrador R0

```

Activities  Terminal  abr 25 09:34  tomas@tomas-Latitude-3420: ~/LabDig/E4
Register group: general
r0      0x37      55      r1      0x0      0      r2      0x15      21
r3      0x22      34      r4      0x37      55      r5      0x30028    196648
r6      0x2ff14    196372  r7      0x3ffff048  1073737800  r8      0x103ec    66540
r9      0x3fffd78  1073737080  r10     0x0      0      r11     0x2ff14    196372
r12     0x3ffa0000  1073348608  sp      0x407ffe00  0x407ffe00  lr      0x3fe2e3f8  1071834104
pc      0x10430    0x10430 <fim>
fpsidr  0x410430f0  1090793712  fpexc   0x40000000  1073741824  AFSR0_EL1  0x0      0
AFSR0_EL1  0x0      0      DBGDIIDR  0x3515f021  890630177  DBGDSAR  0x0      0
DBGGBVR  0x0      0      DBGBCR    0x0      0      DBGWVR    0x0      0
DBGWCR    0x0      0      PAR       0x0      0      DBGGBVR    0x0      0

item-4.5.6.s
19      bne     loop      @ se contador !=1, para, se não continua
20      b       fim
21
22      fim:
B+ 23      mov     r0, r4      @ resultado final em r0
> 24      swi     0x0         @ resultado final em r4
25
26
27
28      .data
29      fibonaci:

remote Thread 1.64072 In: fim
(gdb) b fim
Breakpoint 1 at 0x1042c: file item-4.5.6.s, line 23.
(gdb) c
Continuing.

Breakpoint 1, fim () at item-4.5.6.s:23
(gdb) s
(gdb) p/d $r0
$1 = 55
(gdb)

```