PCS3432 - Laboratório de Processadores

Relatório 4

Nome	NUSP
Victor Hugo Chimenez Queiroz	11288405
Jong Hwan Hong	11261083

Prof: Wilian França Costa/Marco Túlio Carvalho de Andrade

Exercícios do capítulo 4

Exercício 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.

Nesse exercício, precisamos basicamente utilizar a instrução LDR de forma pré-indexada e pós-indexada.

```
.text
.global main

main:
        ADR r2, array           @ carregamos a posicao inicial do array em r2
        MOV r4, #5              @ carregamos o indice do vetor que queremos acessar
n = 5

        MOV r1, #2              @ carregamos um valor qualquer para y
        MOV r5, #7              @ carregamos o r5 que servira para colocar um valor
no array[5]
        STR r5, [r2, r4, LSL #2]   @ salvamos o valor de r5 no array[5]

preindexado:
        LDR r3, [r2, r4, LSL #2]   @ carregamos em r3 o valor guardado no endereco
de indice 5 do array
        ADD r0, r1, r3          @ realizamos a soma pedida, r0 = r1 + r3 equivale a x
= y + array[5]

posindexado:
        ADD r2, r4, LSL #2        @ atualizamos o indice do array antecipadamente
        LDR r3, [r2]            @ realizamos um LDR pos indexado do valor pegado do
endereco atualizado
        ADD r0, r1, r3          @ realizamos a soma

        SWI 0x0                 @ fim do programa

array: .space 100
```

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.

Usando a forma pré-indexada:

```
@ carrega o valor armazenado em array[5] em r0
LDR r0, [r2, #20]
@ adiciona o valor de y ao valor armazenado em r0
ADD r0, r0, r1
@ armazena o valor atualizado em array[10]
STR r0, [r2, #40]
```

Usando a forma pós-indexada:

```
@ carrega o valor armazenado em array[5] em r0 e atualiza o
index do array
ADD r2, r2, #20
LDR r0, [r2], #20
@ adiciona o valor de y ao valor armazenado em r0
ADD r0, r0, r1
@ armazena o valor atualizado em array[10]
STR r0, [r2]
```

4.5.3 Array assignment

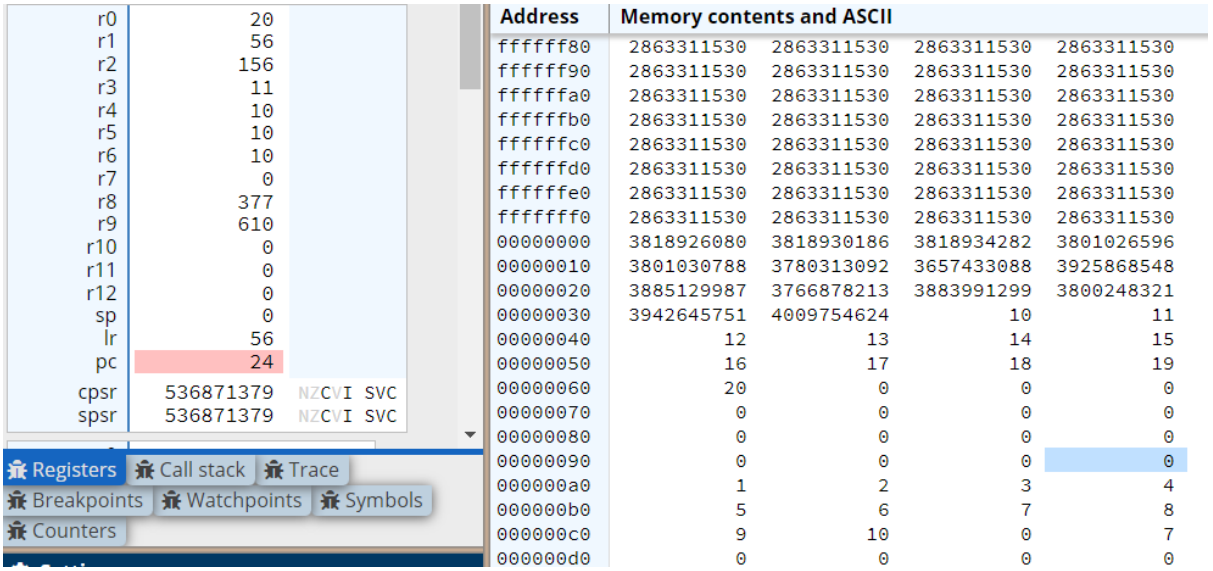
Write ARM assembly to perform the following array assignment in C:

```
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.

```
.text
.global main

main:
        MOV r3, #0              @ index i = 0
        MOV r4, #10             @ valor de index final 10
        MOV r5, #10             @ variavel c = 10
        ADR r1, a               @ carregamos a posicao inicial do a em r1
        ADR r2, b               @ carregamos a posicao inicial de b em r2

check:
        CMP r3, r4                      @ checamos se i<=10
        BLE loop                @ se i <= 10, vamos para o loop principal
        B   fim                 @ caso contrario, finalizamos

loop:
        LDR r6, [r2, r3, LSL #2]   @ carregamos o r6 com o valor guardado em b[i]
        ADD r0, r6, r5          @ realizamos a soma b[i] + c
        STR r0, [r1, r3, LSL #2]   @ guardamos o resultado da soma em a[i]
        ADD r3, r3, #1          @ incrementamos o i
        B   check               @ voltamos para a checagem

fim:
        SWI 0x0                 @ fim do programa


a: .space 100
b: .word  0,1,2,3,4,5,6,7,8,9,10
```

No código acima, realizamos o loop descrito, com o array b carregado com os valores de 0 a 10 e c = 10. Obtivemos um resultado esperado, de 10 a 20 carregados no array a.



r0	20
r1	56
r2	156
r3	11
r4	10
r5	10
r6	10
r7	0
r8	377
r9	610
r10	0
r11	0
r12	0
sp	0
lr	56
pc	24
cpsr	536871379 NZCVI SVC
spsr	536871379 NZCVI SVC

Registers | Call stack | Trace | Breakpoints | Watchpoints | Symbols | Counters

Address	Memory contents and ASCII			
ffffff80	2863311530	2863311530	2863311530	2863311530
ffffff90	2863311530	2863311530	2863311530	2863311530
ffffffa0	2863311530	2863311530	2863311530	2863311530
ffffffb0	2863311530	2863311530	2863311530	2863311530
ffffffc0	2863311530	2863311530	2863311530	2863311530
ffffffd0	2863311530	2863311530	2863311530	2863311530
ffffffe0	2863311530	2863311530	2863311530	2863311530
fffffff0	2863311530	2863311530	2863311530	2863311530
00000000	3818926080	3818930186	3818934282	3801026596
00000010	3801030788	3780313092	3657433088	3925868548
00000020	3885129987	3766878213	3883991299	3800248321
00000030	3942645751	4009754624	10	11
00000040	12	13	14	15
00000050	16	17	18	19
00000060	20	0	0	0
00000070	0	0	0	0
00000080	0	0	0	0
00000090	0	0	0	0
000000a0	1	2	3	4
000000b0	5	6	7	8
000000c0	9	10	0	7
000000d0	0	0	0	0

Na imagem acima observamos os registradores e a memória após a execução do código. Observamos pelos valores dos registradores r1 e r2 os endereços de a e b respectivamente, sendo r1 -> 56 ou 0x38; e r2 -> 156 ou 0x9C. Observando os valores guardados em decimal nesses endereços, observa-se a sequência de valores esperados para cada array.

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 *a, 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.

a) Índices

```
.text
.global main

main:
        ADR r1, array           @ achamos o endereco do array
        MOV r3, #0              @ iniciamos i = 0
        ADR r4, s               @ achamos o endereco de s
        LDR r2, [r4]            @ carregamos o valor passado de s em r2
        MOV r0, #0              @ r0 sera o registrador para o numero 0

loop:
        STR r0, [r1, r3, LSL #2]   @ salvamos o 0 no array[i]
        ADD r3, r3, #1          @ incrementamos o i
        CMP r3, r2              @ verificamos se i<s
        BLT loop                @ se for, volta para o loop, se nao passa e termina o
codigo

fim:
        SWI 0x0                 @ fim do programa


array: .space 100
s:     .word 5
```

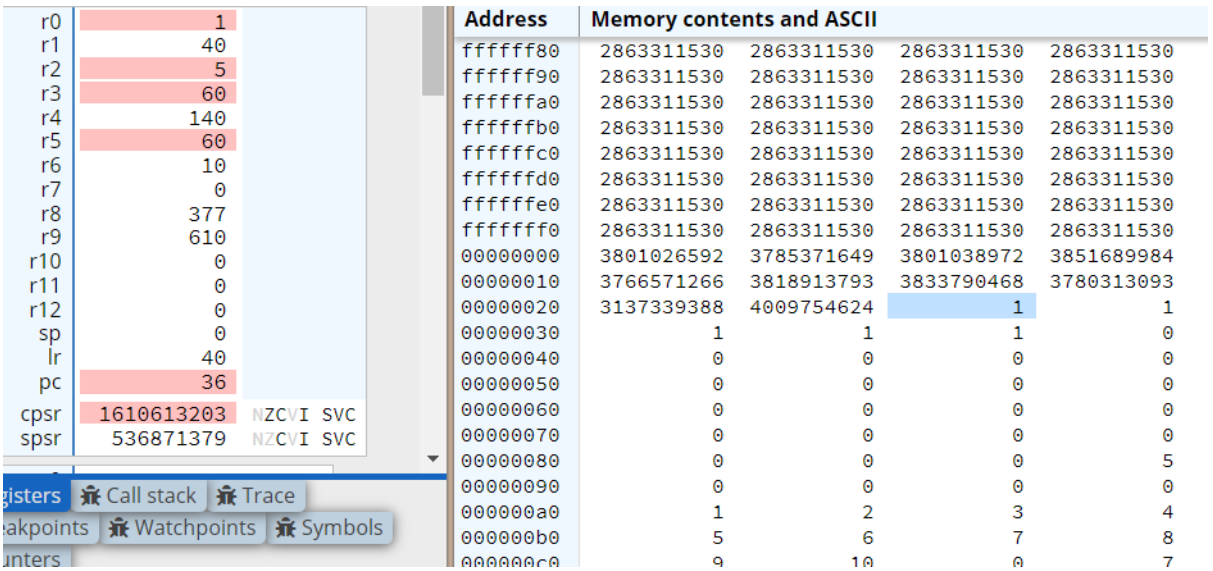
Aqui estamos inicializando o array com zeros até o índice 4 (i < 5), s pode ser mudado pela última linha do código.

b) Ponteiros

```
@ ex 4.5.4 b) do livro
@ Escreva o código assembly ARM para realizar uma inicializacao de um array com
zeros
@ nusp 11261083


.text
.global main

main:
```

```
        ADR r1, array           @ achamos o endereco do array
        MOV r3, r1              @ p (r3) recebe o endereco do array[0]

        ADR r4, s               @ achamos endereco de s
        LDR r2, [r4]            @ carregamos em r2 o valor de s
        ADD r5, r1, r2, LSL #2  @ calculamos o endereco do array[s]

        MOV r0, #0              @ r0 sera o registrador que segura o 0

loop:
        STR r0, [r3], #4        @ carregamos o valor r0 = 0 nos endereco apontado pelo *p (r3), e pelo modo pos indexado, incrementamos 1 no endereco (4 bytes de espacamento)
        CMP r3, r5              @ verificamos se ja atingimos o endereco alvo &array[s] salvo no r5
        BLT loop                @ se ainda nao, voltamos para o loop. se sim, passa para o fim do programa


fim:
        SWI 0x0                 @ fim do programa


array: .space 100
s:     .word 5
```

Este código realiza a mesma coisa de cima, porém ao invés de utilizar um índice i para calcular o endereço do array[i], utilizamos a variável p para servir de ponteiro para o array, logo p (r3) no início contém o valor de &array[0], e no final contém o valor de &array[s].



Um teste rápido foi realizado com s = 5 e, ao invés de inicializar com 0 que foi mais difícil de enxergar, inicializamos com o valor 1.

4.5.5 and 4.5.6 The Fibonacci sequence

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.

Ao invés dos 12 primeiros números, foi pedido achar até o índice sendo os últimos 2 valores do NUSP. Como o final do NUSP utilizado foi de 05, a instrução era utilizar 05 + 10 = 15.

```
.text
.global main

main:
        MOV r0, #15       @ r0 guarda o indice alvo n = 15
        ADR r1, array     @ carrega o endereco do array (de fibonacci) no registrador r1 (f(0))
        MOV r8, #0        @ r8 servira de registrador temporario, nesse caso carrega o primeiro valor da sequencia para inicializar (f(0) = 0)
        STR r8, [r1, #0]  @ salvamos 0 na primeira posicao da sequencia
        MOV r9, #1        @ carregamos o segundo valor para inicializar (f(1) = 1)
        STR r9, [r1, #4]  @ salvamos 1 na segunda posicao da sequencia, que seria +4 da posicao 0 pois sao 4 bytes de espacamento cada elemento
```

```
        MOV r2, #2          @ indice n inicial

check:
        CMP r2, r0          @ verificacao para o branch
        BLE pronto          @ se n <= 15 passamos para o loop pronto

        B   fim             @ caso n passe de 15, termina o programa

pronto:
        ADD r3, r8, r9              @ somamos os ultimos 2 valores da sequencia (f(n) = f(n-1) +
f(n-2))
        MOV r8, r9                  @ atualizamos os registradores temporarios para os valores
novos de f(n-2)
        MOV r9, r3                  @ e f(n-1)
        STR r3, [r1, r2, LSL #2]    @ salvamos o valor calculado da iteracao atual para a
posicao de memoria r1 + 4*n, pois como explicado anteriormente, sao 4 bytes de
espacamento entre os elementos do array
        ADD r2, r2, #1              @ incremento do indice
        B   check                   @ voltamos para a verificacao


fim:
        ADR r4, ultimo      @ carregamos o endereco do label ultimo, que guardara o valor
final da sequencia
        STR r3, [r4]        @ salvamos o valor final ao endereco do label ultimo
        SWI 0x0             @ fim do programa


array: .space 400
ultimo: .word 0
```

Prints dos resultados estão na entrega E4 da aula.