Tarefa 4 - Load and Store

Carlos Eduardo Sena Rebouças Garcia 9345255

Diego Boccoli Gallego 9344744

Gabriel Rocha Amorim 9395112

**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.**

Pós indexado:

.text

.globl main

main:

LDR r2, =arr

ADD r4, r2, #5

LDR r3, [r4]

ADD r0, r1, r3

B out

arr:

.word 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3

out:

Pré indexado:

.text

.globl main

main:

LDR r2, =arr

LDR r3, [r4, #5]

ADD r0, r1, r3

B out

arr:

.word 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3, 0x0, 0x1, 0x2, 0x3

out:

**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.**

Pós indexado

MOV r2, =array

ADD r2, r2, #20

LDR r3, [r2], #20

ADD r3, r3, r1

STR r3, [r2]

Pré indexado:

LDR r3, [r2, #20]!

ADD r3, r3, r1

STR r3, [r2, #20]

**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.**

loop:CMP r3, #40

BGT out

LDR r5, [r2, r3]

ADD r5, r5, r4

STR r5, [r1, r3]

ADD r3, r3, #4

B loop

out:

**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)

MOV r3, #0

loop:CMP r2, r3

BLT out

MOV r5, #0

STR r5, [r1, r3]

ADD r3, r3, #4

B loop

out:

b)

MOV r3, r1

MLA r4, r2, #4, r1

loop:CMP r4, r3

BLT out

MOV r5, #0

STR r5, [r3], #4

B loop

out:

**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.**

.text

.globl main

main:

LDR r0, =fibon

LDR r7, =fibon + 13

loop: CMP r7, r0

BLE out

LDRB r1, [r0], #1

LDRB r2, [r0]

ADD r2, r2, r1

STRB r2, [r0, #1]

B loop

fibon:

.byte 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

out:

**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).**

.text

.globl main

main:

MOV r1, #9

SUB r1, r1, #2

MOV r0, #1

MOV r2, #0

loop: CMP r1, #0

BLE out

ADD r2, r0, r2

ADD r0, r0, r2

SUB r1, r1, #2

B loop

out: CMP r1, #-1

MOVEQ r0, r2