Tarefa 5 - Conditional Execution and Loops

Carlos Eduardo Sena Rebouças Garcia 9345255

Diego Boccoli Gallego 9344744

Gabriel Rocha Amorim 9395112

**5.5.1 For loop**

**Code the following C code in assembly. The arrays a and b are located in memory at 0x4000 and 0x5000 respectively. You may wish to type your code into the assembler to check for syntax.**

**for (i=0; i < 8; i++) {**

**a[i] = b[7-i];**

**}**

MOV r0, #0

loop: CMP r0, #32

BGE out

RSB r1, r0, #28

LDR r2, [r1, b]

STR r2, [r0, a]

ADD r0, r0, #4

B loop

out: SWI =0x0

a:

.word 1, 2, 3, 4, 5, 6, 7, 8

b:

.word 1, 2, 3, 4, 5, 6, 7, 8

**5.5.2 Factorial calculation**

**To take advantage of the idea of conditional execution, let's examine the algorithm for computing n!, where n is an integer, defined as:**

**For a given value of n, the algorithm iteratively multiplies a current product by a number one less than the number it used in the previous multiplication. The code would continue to loop until it is no longer necessary to perform a multiplication, first by subtracting one from the next multiplier value and stopping if it is equal to zero. Here we can use the concepts of:**

**• conditional execution to conditionally perform the multiplication**

**• saving of the current product into a temporary register**

**• branch back to the start of the loop.**

**In writing routines that have loops and branches, many programmers start with a nonzero value and count down, rather than up, because you can use the Z flag to quickly determine whether the loop count has been exhausted.**

**Fill in the blanks in the following code segment. Then run the code on the evaluation board by including the necessary header information and compiler directives. Using a starting value of 10 for n, demonstrate the result to your lab instructor and print out the register bank before and after the program runs.**

**factorial MOV r6,#0xA ; load 10 into r6**

**MOV r4,r6 ; copy n into a temp register loop**

**SUBS \_\_\_\_\_\_\_\_\_\_ ; decrement next multiplier**

MOV r6, #0xA

MOV r0, #0x1

factorial: CMP r6, #0

BEQ end

MUL r0, r0, r6

SUB r6, r6, #1

B factorial

end:

**5.5.3 Find maximum value**

**In this exercise, you are to find the largest integer in a series of 32-bit unsigned integers. The length of the series is determined by the value in register r5. The maximum value is stored in the memory location 0x5000 at the end of the routine. The data values begin at memory location 0x5006. Choose 11 or more integers to use. Use as much conditional execution as possible when writing the code. Demonstrate the program to your lab instructor and print out the memory space starting at 0x5000 before and after the program runs. Be sure to include enough memory space to show all of your 32-bit integer values.**

LDR r0, #0x0

MOV r1, #0x0

MOV r5, #76

loop: LDR r2, [r1], #4

CMP r0, r2

MOVLT r3, r2

CMP r1, r5

BGE out

B loop

maxValue

.byte 0x0

arr:

.byte 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xDDDD,

0xAAAA, 0xFFFF, 0x0111, 0x1000

out: STR r3, =maxValue

**5.5.4 Finite state machines: a non resetting sequence recognizer**

**1. Consider an FSM with one input X and one output Z. The FSM asserts its output Z when it recognizes an input bit sequence of b1011. The machine keeps checking for the sequence and does not reset when it recognizes the sequence. Here is an example input string X and its output Z:**

**X = …0010110110…**

**Z = …0000010010…**

**Write ARM assembly to implement the sequence recognizer. Start with the initial input X in r1. Finish with the output Z in r2 at the end of the program.**

MOV r9, #1

MOV r2, #0

MOV r5, #0

LDR r1, =0b0010110110

loop: CMP r5, #28 @ Compara se já verificou os 32 bits

BGT out @ Saída do loop

AND r0, r1, #0xF @ Pega os bits de teste

CMP r0, #0b1011 @ Verifica condição da FSM

ADDEQ r2, r2, r9, LSL r5 @ r2 = r2 + (1 >> r5)

MOV r1, r1, LSR #1 @ Movimenta registrador de entrada

ADD r5, r5, #1 @ Incrementa índice

B loop @ Fecha loop

out:

**2. Now write the code to recognize any sequence Y up to 32 bits. Start with the recognizing sequence Y in r8 and the size of Y in r9. For example, to recognize the sequence Y = b0110110, then r8 = 0x36 and r9 = 0x7 before program execution. Everything else should be the same is in Step 1. Make sure that your program works for every case, including the case when r9 = 1 or r9 = 32.**

MOV r7, #1

MOV r6, #1

MOV r2, #0

LDR r1, =0b10110110

LDR r8, =0b10110110

MOV r9, #8

MOV r5, #0

RSB r4, r9, #32

MOV r7, r7, LSL r9

SUB r7, r7, #1

loop: AND r0, r1, r7

CMP r0, r8

ADDEQ r2, r2, r6, LSL r5

MOV r1, r1, LSR #1

ADD r5, r5, #1

CMP r5, r4

BGE out

B loop

out:

**5.5.5 Sequential parity checker**

**Write ARM assembly to inspect the parity of a value initially held in r0. If r0 has an odd number of ones, the program ends with 0x0001 in r1. If r0 has an even number of ones, the program ends with 0x0000 in r1.**

LDR r0, =0x586

LDR r1, =0x0

loop: CMP r0, #0

BEQ out

AND r2, r0, #1

ADD r1, r1, r2

MOV r0, r0, LSR #1

B loop

out: AND r1, r1, #1

SWI 0x123456