ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Graduação em Engenharia Computação

PCS3732 - Laboratório de Processadores

Professor Jorge Kinoshita

Grupo 10 - Planejamento E5

Arthur Pires da Fonseca NUSP: 10773096

Sumário

5.1.2 B. Responda:

1. Traduza as seguintes instruções em uma única instrução ARM:

a. Adicione registradores r3 e r6 somente se N = 0 (N estah "clear"). Armazene o resultado no registrador r7.

ADDPL R7, R3, R6

b. Adicione registradores r3 e r6 somente se N = 1. Armazene o resultado no registrador r7.

ADDMI R7, R3, R6

c. Multiplique os registradores r7 e r12, colocando os resultados no registrador r3 somente se C está setado (C = 1) e Z = 0.

MULHI R3, R7, R12

d. Multiplique os registradores r7 e r12, colocando os resultados no registrador r3 somente se C clear ou Z set.

C clear => unsigned <

Z set => equal to

(C clear) or (Z set) => unsigned less than or equal to

MULLS R3, R7, R12

e. Compare os registradores r6 e r8 somente se Z está zerado.

CMPNE R6, R8

f. Compare os registradores r6 e r8 somente se Z set ou N ≠ V

Z set => equal to

N ≠ V => less than

(Z set) or N ≠ V => less than or equal to

CMPLE R6, R8

2. Observe a seguinte função em C:

```
int foo(int x, int y) {
  if ((x + y) >= 0)
      return 0;
  else
      return 1;
}
```

Suponha que ela tenha sido compilada e traduzida no seguinte código:

```
foo      ADDS r0,r0,r1
         BPL PosOrZ
done     MOV r0, #0
         MOV pc, lr
PosOrZ   MOV r0,#1
         B done
```

O compilador gerou o código corretamente? O compilador retorna 0 ou 1 em r0. Se não está bom o código, corrija. Altere o código para que ele execute a função em somente 4 instruções (dica: use execução condicional).

O código original está incorreto, porque ao fim da função sempre seta r0 para zero.

Correção + otimização de quantidade de instruções:

```
foo:    ADDS r0,r0,r1
        MOVPL r0, #0  @ r0 = 0 se (x + y) >= 0
        MOVMI r0, #1  @ r0 = 1 se (x + y) < 0
        MOV pc, lr
```

5.1.3 C. IMPORTANTE: Implemente e teste o codigo de 5.5.4.1

5.5.4 Finite state machines: a non resetting sequence recognizer

Exercicio 5.5.4 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...

1. **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.**

```
@ Exercicio 5.5.4.1 do livro
@ Para debugar este codigo:
@ gcc fsm.s && gdb a.out
        .text
        .globl   main
main:
        ldr r1, =0xb0b0b0b0   @ input X

        mov r2, #0            @ output Z

        ldr r3, =0xb          @ pattern

        mov r4, #1            @ aux = 1

        mov r0, #0            @ last_time = false
        ldr r8, =0x80000000   @ limit pattern


loop:
        sub r5, r1, r3        @ X - pattern
        ands r5, r5, r3       @ match mask and hole
        addeq r2, r2, r4      @ if(found pattern) Z += aux

        mov r4, r4, lsl #1    @ aux *= 2
        mov r3, r3, lsl #1    @ pattern shift

        cmp r0, #1
        beq end               @ if(last_time) goto end

        cmp r3, r8            @ pattern big enough?
        movcs r0, #1          @ last_time = true

        b loop
```

```
end:
        swi 0x0
```