Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica
e de Computação

EA871 - Laboratório de Programação Básica de Sistemas Digitais

Roteiro 3 - Linguagem de Montagem (*Assembly*)

Aluno: Fernando Teodoro de Cillo

RA: 197029

Campinas
Abril de 2022

Introdução:

O intuito deste experimento é ensinar programação para o núcleo Cortex-M0+ em linguagem de montagem (*assembly)* utilizando o IDE CodeWarrior. Serão utilizados os mnemônicos dos códigos de máquina do conjunto de instruções THUMB (uma versão reduzida de ARM).

Experimento:

1

1.a

Tabela 1: sequência de mnemônicos aplicada na execução de cada operação aritmética

Operação Aritmética	Mnemônicos	Tipos de Instrução
Adição	ldr r2, [r7, #0]	acesso à memória
	ldr r3, [r7, #4]	acesso à memória
	add r2, r3	lógico-aritmética
	str r2, [r7, #8]	acesso à memória
Subtração	ldr r3, [r7,#4]	acesso à memória
	ldr r2, [r7, #0]	acesso à memória
	sub r2, r3	lógico-aritmética
	str r2, [r7, #12]	acesso à memória
Multiplicação	ldr r3, [r7,#4]	acesso à memória
	ldr r2, [r7, #0]	acesso à memória
	mul r2, r3	lógico-aritmética
	str r2, [r7, #16]	acesso à memória

É importante ressaltar que na tabela 1 a linha `ldr r3, [r7, #4]` aparece em todas as operações, apesar de só ser utilizada uma vez no código `rot3_arit.s`. Como o valor 7 já foi carregado para r3 na primeira vez que a função foi chamada e ele não foi reescrito, não haveria necessidade de o programador incluir esta linha de código novamente, diferente do valor em r2, que é o registrador utilizado tanto para o operando 1 quanto para o resultado das operações e, portanto, deve ser carregado novamente pelo programador antes de realizar as próxims operações. Assim, esse mnemônico é utilizado para todas as operações, pois caso contrário não haveria nenhum valor no operando 2 (r3) e ocorreria erro.

1.b

Ao registrador r7 é assinalado o endereço 0x20002fdc pela linha `add r7, sp, 0`, como vemos pela aba Registers da figura 1. Os endereços em que são armazenados os operandos e os resultados são todos relativos a r7 (o endereço [r7, #4] representa um deslocamento de 4 *bytes* em relação ao endereço de r7, ou seja, a próxima palavra da memória) e, portanto, na aba Memory focada no endereço de r7 é possível verificar o valor armazenado de cada um deles, como mostra a figura 2.

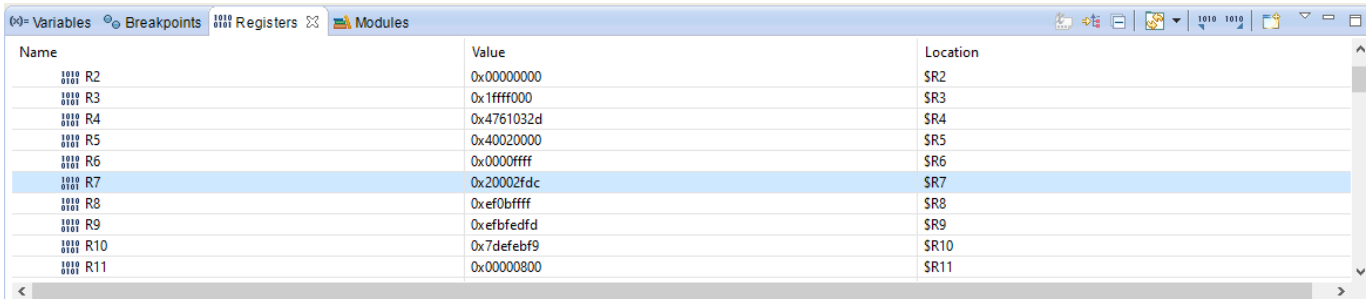


Figura 1: endereço dos registradores

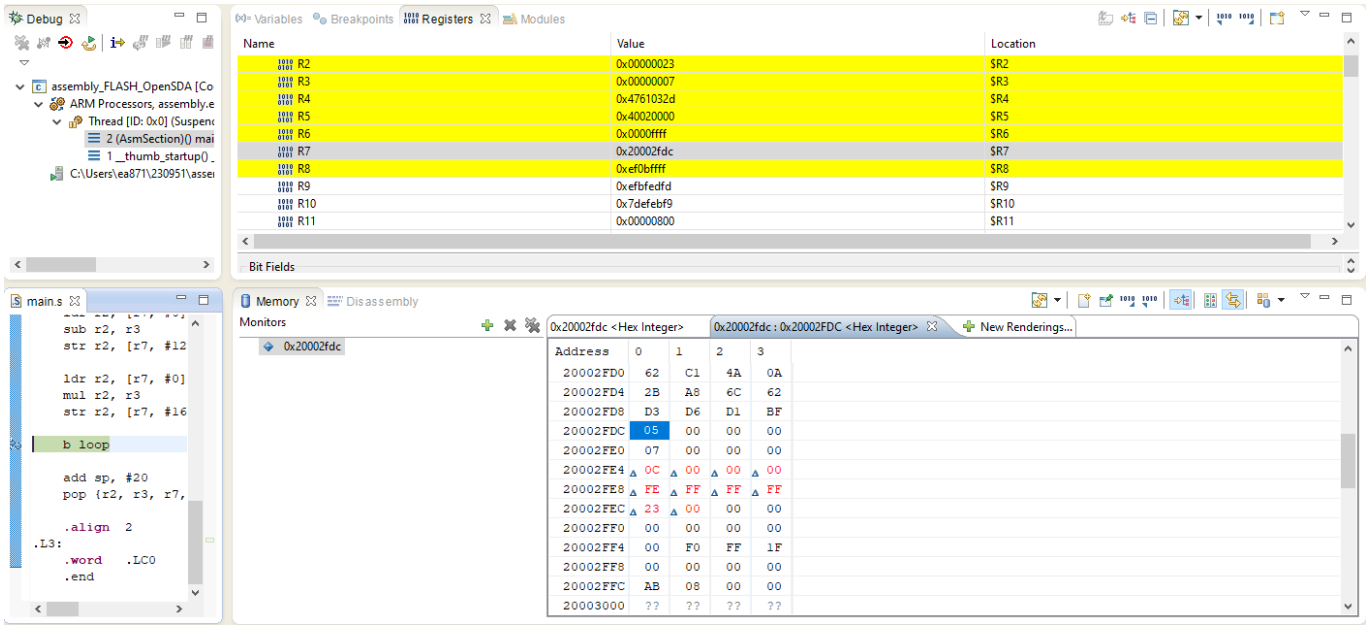


Figura 2: aba *Memory* com os valores dos registradores

1.c

O registrador r7 é o registrador base, usado como referência para os outros endereços. Já r3 é utilizado para o operando 2 e r1 armazena tanto o operando 1 quanto o resultado das operações.

1.d

A ideia de alterar o valor do *sp* é não sobrescrever os valores da pilha (a menos que essa seja a intenção). Lembrando que a pilha é armazenada de baixo para cima (endereços maiores primeiro e menores depois), a operação `sub sp, #20` "sobe"o contador em 5 palavras porque as 4 primeiras já estão escritas. Analogamente, `add sp, #20` "desce"o contador, porque os valores que estavam na pilha não mais nos interessam (mas essa função não os apaga).

Tabela 2: modos de endereçamento

	Operando 1	Operando 2
`sub sp, #20`	direto por registrador	imediato
`add r2, r3`	direto por registrador	direto por registrador
`ldr r3, [r7, #4]`	direto por registrador	indireto por registrador
`b loop`	absoluto	-

1.e

2

2.a

- mov r2, #5: move (armazena) o valor #5 em r2;
- and r2, r3: operação lógia AND bit-a-bit entre os bits de r2 e r3;
- asr r2, r3: desloca os bits de r2 em r3 casas para a direita e escreve nos bits que foram deslocados o valor do segundo bit mais significativo (ignora o bit de sinal);
- eor r2, r3: operação lógia OR-exclusiva bit-a-bit entre os bits de r2 e r3;
- lsl r2, r3: desloca os bits de r2 em r3 casas para a esquerda e escreve nos bits que foram deslocados o valor zero;
- lsr r2, r3: desloca os bits de r2 em r3 casas para a direita e escreve nos bits que foram deslocados o valor zero;
- mvn r2, r2: move o valor de r2 para r2 e o nega bit-a-bit;
- neg r2, r2: faz o complemento de 2 do valor de r2 e armazena em r2;
- ror r2, r2: rotaciona para a direita o valor de r2 (desloca para a direita mas não descarta os bits, e sim os armazena à esquerda);
- add sp, #40: adiciona o valor 40 (bytes) ao ponteiro da pilha.

2.b

O operando de `b` é a *label* (rótulo) do bloco de instruções (usualmente um laço) para onde o contador de programa será direcionado. Essa operação não é chamada indicando o endereço da instrução seguinte, o laço, e sim o seu rótulo, que deve ser declarado pelo programador. No arquivo `rot3_bits.s` a instrução `b loop` faz o PC apontar para a primeira instrução dentro do rótulo `loop`, ou seja, `ldr r2, [r7, #0]`, na linha 31.

Para a instrução `b loop`, a codificação seria 1101111000011111 (cond = 1110 indica que não há condição e imm8 = 00011111 indica que o desvio deve ocorrer para a linha 31).

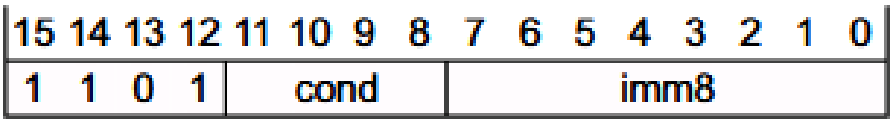


Figura 3: Codificação da instrução `b`

2.c

As instruções `POP` e `PUSH` carregam (*load*) valores do topo da pilha para registradores e salvam (*store*) valores de registradores no topo da pilha, respectivamente. A linha de código `push r2, r3, r7, lr`, por exemplo, salva no topo da pilha os valores dos registradores r2, r3, r7 e lr, nessa ordem.

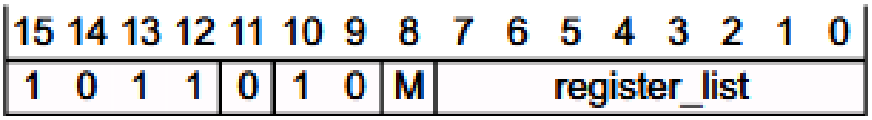


Figura 4: Codificação da instrução `push`

Para a instrução `push {r2, r3, r7, lr}`, a codificação seria 1011010110001100 (M=1 indica modo de operação THUMB e register_list 10001100 tem 1s nos registradores utilizados).

2.d

A intrução `ldr` carrega o valor armazenado no endereço do registrador base mais o *offset*. `ldr r3, [r7, #4]`, por exemplo, carrega o valor do endereço [r7, #4] no registrador r3. Já a operação `str` faz o contrário, ou seja, `str r2, [r7, #12]` armazena o valor de r2 no endereço [r7, #12].

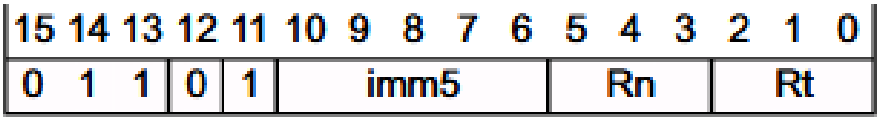


Figura 5: Codificação da instrução `ldr`

Para a instrução `ldr r3, [r7, #4]`, a codificação seria 0110100100111011 e para a instrução `str r2, [r7, #4]`, a codificação seria 0110000100111010 .

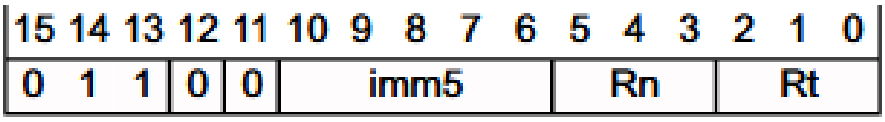


Figura 6: Codificação da instrução `str`

2.e

Para a instrução `pop {r2, r3, r7, pc}`, a codificação seria 1011110110001100 (P=1 indica modo de operação THUMB e register_list 10001100 tem 1s nos registradores utilizados).

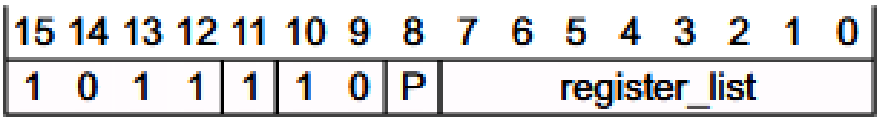


Figura 7: Codificação da instrução `pop`

3

Arquivo *fatorial.s* enviado compactado via moodle.

4

4.a

Os valores armazenados nos registradores, pela figura 8, são:

- r0: 0x00000000 (0 em decimal)
- r1: 0x1ffff000 (536866816 em decimal)
- r2: 0x00000000 (0 em decimal)
- r2: 0x0000003c (60 em decimal)

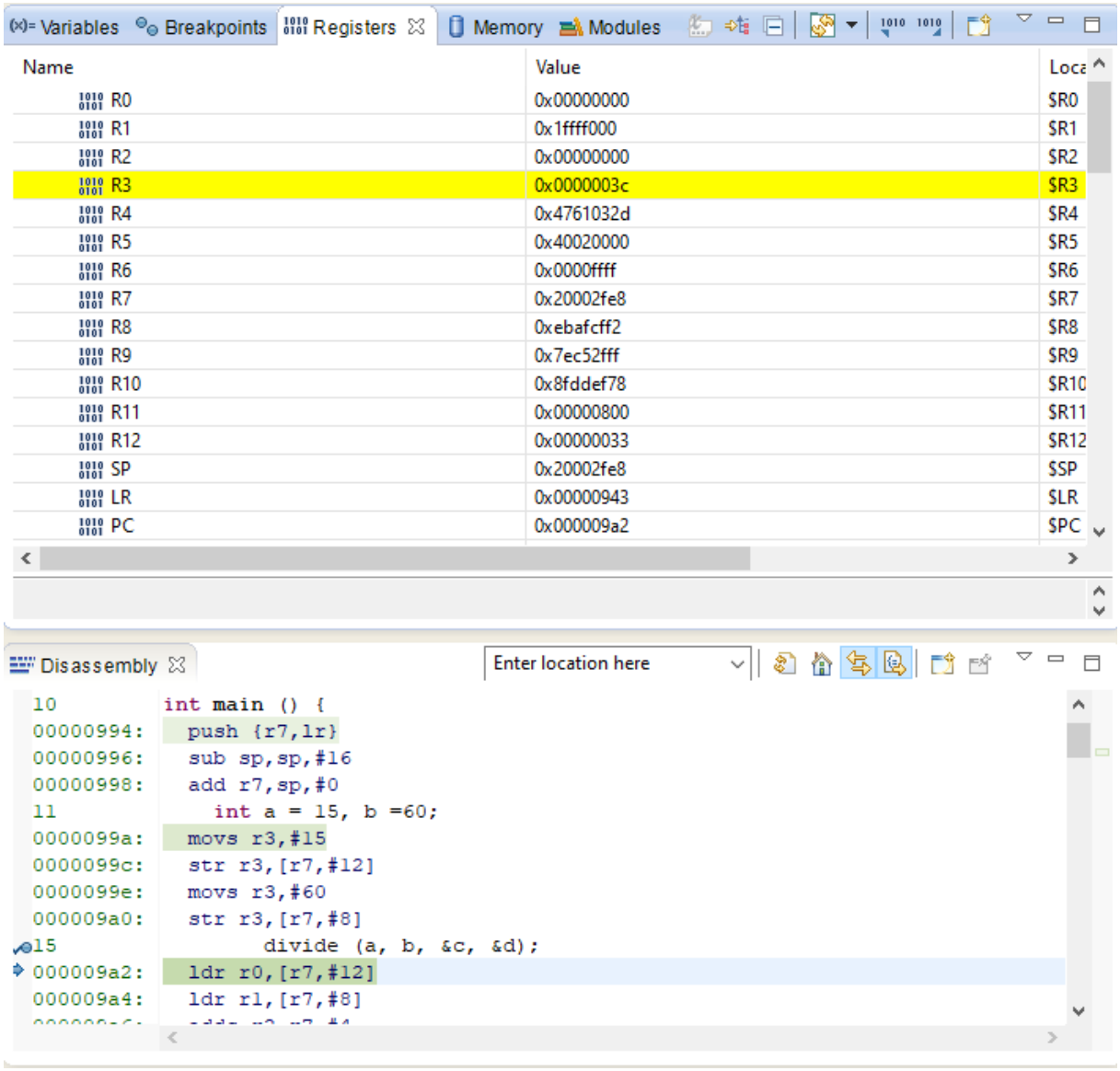


Figura 8: aba Registers

4.b

O valor armazenado no registrador base r7 é 0x20002fe8.

4.c

O valor armazenado em SP também é 0x20002fe8.

4.d

Na figura 9 vemos que os endereçso de memória que armazena as variáveis `a, b, c,` e `d` são as palavras logo acima (valores diminuindo de 4 em 4 *bytes*) do endereço de r7, o registrador base (0x20002ff8). A figura 10 mostra a aba *Memory* e podemos ver que os valores armazenados são exatamente o que se esperava, ou seja, o das variáveis da nossa função.

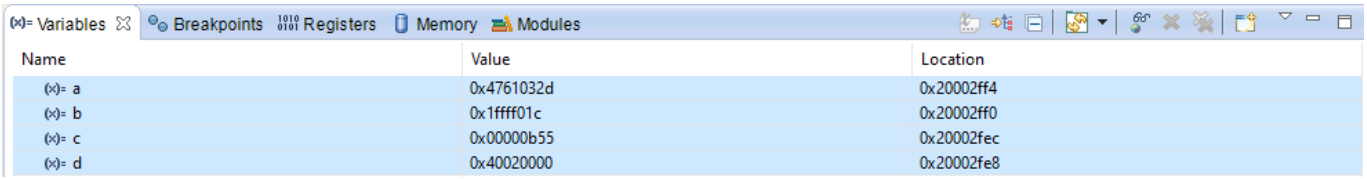


Figura 9: aba Variables

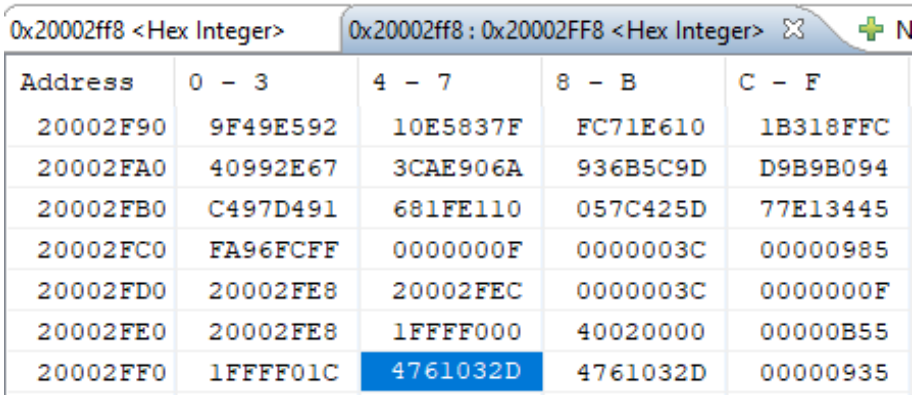


Figura 10: aba Memory

4.e

A instrução `cpy sp, r7` copia o valor de r7 no *Stack Pointer*, a intrução `add sp, sp, #16` faz adiciona o valor #16 no *Stack Pointer* e `pop {r7, pc}` armazena os valores do topo da pilha em r7 e no *Program Counter*.

5

5.a

Conferir tabela 3.

5.b

Conferir tabela 4 e figura 11.

Para estimar o tempo T_E, utilizamos o valor da frequência para calcular o período e o multiplicamos pelo número de ciclos de relógio para obter o tempo (em segundos). Assim, temos

$$T_e = \frac{1}{f} \cdot N_{ciclos} \tag{1}$$

Já o erro percentual é calculado por

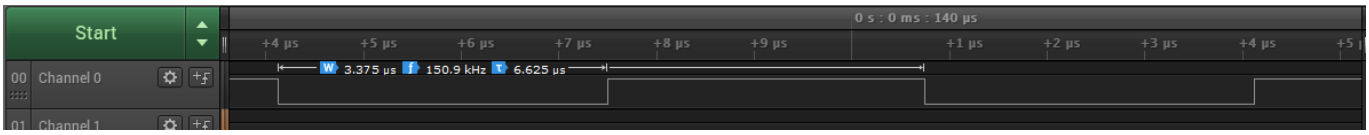
$$erro_{\%} = \frac{|T_e - T_m|}{T_m} \cdot 100 \tag{2}$$

Tabela 3: ciclos de operação por instrução

Instrução	**Ciclos**
delay:	5
push {r2,r3,lr}	4
mov r3, #0	1
iteração:	R0 ciclos
mov r2, #NUM_ITERACOES	1
laço:	6 + #NUM_ITERACOES x 8 x R0
add r3, #0	1
sub r3, #0	1
add r3, #0	1
sub r3, #0	1
add r3, #0	1
sub r2, #1	1
bne laco	2
sub r0, #1	1
bne iteracao	2
pop {r2,r3,pc}	6

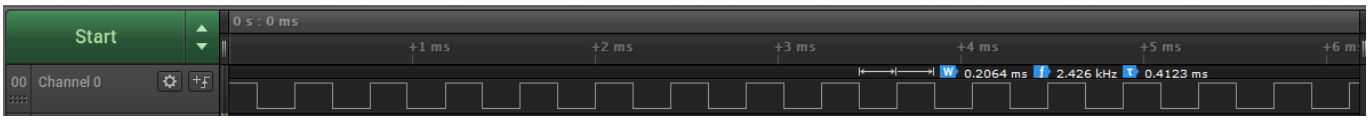
Tabela 4: variação de COUNT

COUNT	Ciclos de relógio	Tempo estimado (f=20971520Hz)	Tempo medido	Erro (%)
1	97	4.6 μs	6.625 μs	30.56%
10	871	41.5 μs	43.5 μs	4.60%
100	8611	0.4106 ms	0.4123 ms	0.48%
1000	86011	4.1013 ms	4.102 ms	0.07%

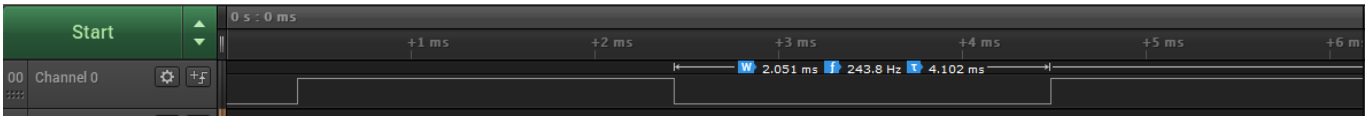


(a) COUNT=1

(b) COUNT=10



(c) COUNT=100



(d) COUNT=100

Figura 11: variação do tempo medido de acordo com COUNT