Exercício avaliativo 1

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB)
Disciplina: Microprocessadores e microcontroladores
Professor: Fagner de Araujo Pereira
Aluno (a): _____

Exercício avaliativo 1 (Peso 100 pontos)

1. Considere um processador no qual há instruções de um operando, que utilizam um único registrador na CPU, conhecido como acumulador. A instrução **LOAD** carrega o operando no acumulador; **MULT** multiplica o operando pelo valor que está no acumulador e armazena o resultado no acumulador; **ADD** realiza a soma do operando com o valor que está no acumulador e armazena o resultado no acumulador; **STORE** armazena o valor do acumulador no operando. Com base nessas instruções de um operando, um engenheiro escreveu o seguinte código:

```
LOAD B
MULT C
ADD D
STORE X
```

Assinale a alternativa que corresponde à operação implementada pelo código descrito:
a) X = B * (C + D)
b) X = (B*C) + D (Essa é a alternativa correta)
c) X = (B + C) * D
d) X = X + D
e) X = B*D + C

2. Considere 2 processadores de uso geral, sendo um de **16 bits** e outro de **32 bits**, operando na mesma frequência de clock de **1.2 GHz**. Os processadores são programados para realizar as seguintes operações:

```
double A = 5.32;
double B = 10.25;
C = A + B;
D= A x B;
```

onde as duas primeiras operações carregam as variáveis do tipo *double* (64 bits) com valores iniciais. Considerando que cada instrução dos processadores é realizada em um único ciclo de clock, exceto a multiplicação, que dura 2 ciclos, quanto tempo é necessário para executar esse conjunto de operações em cada processador?

Para o processador de 16 bits lidar com número de 64 bits, a quantidade de pulsos de clock necessária para cada linha de programa acima é a seguinte:

```
double A = 5.32;        (4 pulsos)
double B = 10.25;       (4 pulsos)
C = A + B;              (4 pulsos)
D= A x B;               (4 pulsos) x 2
                        (total de 20 pulsos)
```

O tempo necessário é então calculado como:

$$tempo_{16} = 20T = 20\frac{1}{f} = 16{,}66ns$$

De maneira análoga, para o processador de 32 bits, temos

```
double A = 5.32;        (2 pulsos)
double B = 10.25;       (2 pulsos)
C = A + B;              (2 pulsos)
D= A x B;               (2 pulsos) x 2
                        (total de 10 pulsos)
```

E o tempo necessário é calculado como:

$$tempo_{32} = 10T = 10\frac{1}{f} = 8{,}33ns$$

3. O trecho de código mostrado abaixo é a transcrição em linguagem Assembly de uma sub-rotina de atraso obtida a partir do processo de compilação para um processador com arquitetura de **16 bits**.

```
delay:  MOV R0, 0x0136  //move 310 para o registrador R0
        CLR R1          //limpa o registrador R1
step:   DJNZ R1, step   //decrementa R1 e desvia para step se for diferente de zero
        DJNZ R0, step   //decrementa R0 e desvia para step se for diferente de zero
        RET             //retorna da sub-rotina
```

Considere que o clock desse processador é de 48 MHz, e que um ciclo de máquina corresponde a quatro ciclos de clock. Considere ainda que, a quantidade de ciclos de máquina necessários para execução de cada instrução é dada conforme a tabela abaixo.

Instrução	Tempo de execução (ciclos de máquina)
`MOV Rn, dado`	1
`CLR Rn`	1
`DJNZ Rn, desvio`	2
`RET`	4

Nessas condições, a sub-rotina gera um atraso de quanto tempo?

Primeiro, vamos contabilizar quantos ciclos de máquina o programa necessita. Nas duas primeiras linhas, temos:

```
delay:  MOV R0, 0x0136  (1 ciclo)
        CLR R1          (1 ciclo)
```

Quando ocorre a primeira execução da linha `DJNZ R1, step`, ocorrerá um underflow em R1, fazendo com que todos os seus bits sejam levados a 1. O teste então verificará que R1 ≠ 0 e o programa desviará para *step*, que é a própria linha que ele está atualmente. Com isso, essa linha será executada repetidamente, até que R1 seja zero novamente. Isso representa 2^{16} execuções, uma vez que o registrador é de 16 bits. Logo:

```
step:   DJNZ R1, step   (2 ciclos) x 2^16
```

A linha seguinte, `DJNZ R0, step`, é executada 1 vez e, até que R0 decresça de 310 até zero, o programa desviará para `step`, repetindo o processo anterior. Isso significa que esse segundo desvio ocorrerá 310 vezes, fazendo com que, em cada vez, o processo anterior seja repetido. Logo:

```
        DJNZ R0, step   (2 ciclos) x 310
```

E a quantidade de ciclos gastos nessas duas linhas é:

$$\{(2 \text{ ciclos}) \text{x } 216\} \text{x } \{(2 \text{ ciclos}) \text{x } 310\} = 81.264.640 \text{ ciclos}$$

Finalmente, na última linha, temos:

```
        RET (4 ciclos)
```

Assim, a quantidade total de ciclos é:

$$Ciclos\_de\_máquina_{totais} = 1 + 1 + 81.264.640 + 4 = 81.264.646\ ciclos$$

Como cada ciclo de máquina ocorre em 4 ciclos de clock, o tempo total de execução da sub-rotina é:

$$tempo = 4\ x\ 81.264.646\ x\ \frac{1}{48M} = 6{,}77s$$

4. Um Datapath é formado por uma ULA capaz de executar 280 operações distintas com até 2 operandos, mais um destino, e um banco com 32 registradores. Considerando apenas o endereçamento dos registradores e a seleção da operação na ULA, quantas linhas de controle possui esse Datapath?

Para o opcode, são necessários:

$$n = \lceil log_2 280 \rceil = 9 \text{ bits}$$

Como o banco possui 32 registradores, são necessários 5 bits para endereçar cada um deles (operandos e destinos). Dessa forma, a quantidade de linhas de controle é igual a:

$$linhas\_de\_controle = 9 + 3\ x\ 5 = 24$$

Exercício avaliativo 1

5. Abaixo, é mostrado o conteúdo inicial (representado no formato decimal) do banco de registradores da arquitetura ARM Cortex M em um determinado instante.

Conteúdo inicial do banco de registradores

Registrador	Conteúdo	Conteúdo final
R15 (PC)	2800	2848
R14 (LR)	0	0
R13 (SP)	1252	1256
R12	250	-40
R11	250	-30
R10	0	75
R9	0	1248
R8	815	990
R7	2	0
R6	0	45
R5	900	900
R4	350	350
R3	0	60
R2	0	5
R1	250	12
R0	100	4

A tabela abaixo apresenta os valores armazenados (em decimal) em uma região da memória RAM nesse mesmo instante.

Memória RAM

Endereço	Endereço	Conteúdo	Conteúdo final
1268	0x04F4	-200	-200
1264	0x04F0	80	80
1260	0x04EC	-40	900
1256	0x04E8	-30	350
1252	0x04E4	75	75
1248	0x04E0	500	990

Sabendo que a pilha de memória da arquitetura ARM Cortex M é descendente, isto é, ao inserir um item na pilha, o ponteiro de pilha é decrementado, determine o conteúdo final do banco de registradores e da região de memória especificada após a execução do código abaixo, que é iniciado no instante referido.

```
início:     MOV R9, #1248
            MOV R1, R0, LSR #3
            MOV R2, #5
            MUL R3, R2, R1
            POP {R10-R12}
teste:      CBZ R7, final
            SDIV R0, R2
            SUB R7, #1
            B teste
final:      ADD R6, R10, R11
            PUSH {R4, R5}
            ORR R8, R4, R5
            STR R8, [R9]
```