

ARQ1 \_ Aula\_16

Tema: Introdução à linguagem de máquina (*assembly*)

Orientação geral:

Apresentar todas as soluções em apenas um arquivo com formato texto (.txt).  
Sugere-se usar como nome Guia\_xx.txt, onde xx indicará o guia, exemplo Guia\_01.txt.  
Todos os arquivos deverão conter identificações iniciais com o nome e matrícula, no caso de programas, usar comentários.  
As implementações e testes dos exemplos em Verilog (.v) fornecidos como pontos de partida, também fazem parte da atividade e deverão ter os códigos fontes entregues **separadamente**, com o código fonte e os módulos de testes, a fim de que possam ser compilados e verificados.  
Sugere-se usar como nomes Guia\_01yy.v, onde yy indicará a questão, exemplo Guia\_0101.v  
As saídas de resultados, opcionalmente, poderão ser copiadas ao final do código, em comentários.  
Quaisquer outras anotações, observações ou comentários poderão ser colocadas em arquivo texto (README.txt) acompanhando a entrega.

Outras formas de solução serão **opcionais**; não servirão para substituir as atividades a serem avaliadas. Caso entregues, poderão contar apenas como atividades extras.  
Os programas com funções desenvolvidas em C, Java ou Python (c, .java, py), como os modelos usados para verificação automática de testes das respostas; caso entregues, também deverão estar em arquivos **separados**, com o código fonte e os módulos de testes, a fim de que possam ser compilados e verificados.  
As execuções deverão, preferencialmente, serão testadas mediante uso de redirecionamento de entradas e saídas padrões, cujos dados/resultados deverão ser armazenados em arquivos textos.  
Os resultados poderão ser anexados ao código, ao final, como comentários  
Planilhas, caso venham a ser solicitadas, deverão ser **programadas** e/ou usar funções nativas.  
Serão suplementares e opcionais, e deverão ser entregues em formato texto, preferencialmente, com colunas separadas por tabulações ou no formato (.csv), acompanhando a solução em texto.  
Arquivos em formato (.pdf), fotos, cópias de tela ou soluções manuscritas também poderão ser aceitos como recursos suplementares para visualização, mas não servirão como substitutos e **não** terão validade para fins de avaliação.  
Os *layouts* de circuitos deverão ser entregues no formato (.circ), identificados internamente.  
Os *layouts* de diagramas deverão ser entregues no formato (.jff), identificados externamente.  
Figuras exportadas pela ferramenta serão aceitas apenas como arquivos para visualização, mas **não** terão validade para fins de avaliação. Separar versões completas (a) e simplificadas (b).

## Atividade: Arquitetura de Computador – Intel 8085

Todos os programas deverão ser testados em simulador.

01.) Dado o exemplo abaixo:

```
// Guia_1600a.txt
// 999999 - Xxx Yyy Zzz
//
// Addition of two 8bit numbers calling add function
// Manually store 1st number in the memory location 0050h
// Manually store 2nd number in the memory location 0051h
// Result is stored in 0052h
//
// F_ADD - Function ADD // A = add (B, C)
// @return A
// @param B
// @param C
//
// JMP MAIN // function area detour
// function ADD( ) {
F_ADD: LDA A,00 // A = 0 // return value
// parameter passing
MOV D,B // D = B // local variable
MOV E,C // E = C // local variable
MOV A,D // A = D
ADD E // A = A+E
RET // return // A
// }
//
// main ( ) {
MAIN: LXI H,0050 // HL = 0050h // dado1
MOV B,M // B = MEM [HL]
INX H // HL = HL+1 // dado2
MOV C,M // C = MEM [HL]
C_ADD: CALL F_ADD // A = F_ADD(B, C)
STA 0052 // MEM[0052] = A
HLT // }
END:

// Area de dados
// dado1: 02h ; primeiro dado em hexadecimal
// dado2: 03h ; segundo dado em hexadecimal
// dado3: 00h ; resultado em hexadecimal
```

02.) Dado o exemplo abaixo:

```
// Guia_1600b.txt
// 999999 - Xxx Yyy Zzz
//
// Addition of two 8bit numbers calling add function using stack
// Manually store 1st number in the memory location 0050h
// Manually store 2nd number in the memory location 0051h
// Result is stored in 0052h
//
// F_ADD - Function ADD // A = add (B, C)
// @return A
// @param B
// @param C
//
// JMP MAIN // function area detour
// function ADD( ) {
F_ADD: POP H // HL = (save) return address
// pop parameters from stack
POP D // DE = BC
MVI A,00 // A = 0 // return value
MOV A,D // A = D
ADD E // A = A+E
PUSH H // HL = (restore) return address
RET // return // A
// }
//
// main ( ) {
MAIN: LXI H,0050 // HL = 0050h // dado1
MOV B,M // B = MEM [HL]
INX H // HL = HL+1 // dado2
MOV C,M // C = MEM [HL]
PUSH B // push parameters into stack
C_ADD: CALL F_ADD // A = F_ADD(B, C)
STA 0052 // MEM[0052] = A
HLT // }
END:

// Area de dados
// dado1: 02h ; primeiro dado em hexadecimal
// dado2: 03h ; segundo dado em hexadecimal
// dado3: 00h ; resultado em hexadecimal
```

03.) Dado o exemplo abaixo:

```
// Guia_1600c.txt
// 999999 - Xxx Yyy Zzz
//
// Extract high nibble of an 8bit number calling function
// Manually store 1st number in the memory location 0050h
// Result is stored in 0052h
//
// F_HI - Function HI    // A = hi(C)
// @return  A
// @param  C
//
        JMP  MAIN    // function area detour
F_HI:    POP  H      // HL = (save)  return address
        // pop parameter from stack
        POP  B      // BC = parameter
        MOV  A,C     // A = C
        ANI  F0     // A = A & F0h
        RAR        // A = A >> 1
        RAR        // A = A >> 1
        RAR        // A = A >> 1
        RAR        // A = A >> 1
        PUSH H      // HL = (restore) return address
        RET         // return // A
        // }
        //
        // main ( ) {
MAIN:    LXI  H,0050 // HL = 0050h // dado1
        MVI  B,00   // B = 0
        MOV  C,M    // C = MEM [HL]
        PUSH B      // push parameter into stack
C_HI:    CALL F_HI   // A = F_HI (C)
        STA  0052   // MEM[0052] = A
        HLT         // }
END:

// Area de dados
// dado1: 24h          ; primeiro  dado em hexadecimal
// dado2: 00h          ; resultado    em hexadecimal
```

## Exercícios

- 01.) Implementar um programa (Guia\_1601.txt) para o processador 8085 para calcular o produto (IMUL) de dois dados positivos com 8 bits cada.

DICA: Usar somas sucessivas e tamanho maior que 8 bits para o resultado.

$$\text{dado03} = \text{dado01} * \text{dado02}$$

- 02.) Implementar um programa (Guia\_1602.txt) para o processador 8085 para calcular o quociente inteiro (IDIV) entre dois dados positivos com 8 bits cada.

DICA: Usar subtrações sucessivas e tamanho maior que 8 bits para o resultado.

$$\text{dado03} = \text{dado01} / \text{dado02}$$

- 03.) Implementar um programa (Guia\_1603.txt) para o processador 8085 para calcular o resto inteiro (IMOD) da divisão entre dois dados positivos com 8 bits cada.

DICA: Usar subtrações sucessivas.

$$\text{dado03} = \text{dado01} \% \text{dado02}$$

- 04.) Implementar um programa (Guia\_1604.txt) para o processador 8085 para encontrar o menor valor de um arranjo.

DICA: Usar o exemplo do BubbleSort, mas não ordenar. Usar a repetição.

- 05.) Implementar um programa (Guia\_1605.txt) para o processador 8085 para converter um valor em BCD (Binary Coded Decimal) para o hexadecimal equivalente.

$$\begin{aligned} \text{DICA: } 24_{(\text{BCD})} &= 24_{(10)} = \text{HI}(24) * 10 + \text{LOW}(24) = 2 * 10 + 4 \\ &= 0000\ 0010_{(2)} * 0000\ 1010_{(2)} + 0000\ 0100_{(2)} = 0001\ 0100_{(2)} + 0001\ 1000_{(2)} = 18_{(16)} \end{aligned}$$

$$\text{dado02} = \text{HI}(\text{dado01}) * 10 + \text{LOW}(\text{dado02})$$

## Extras

- 06.) Implementar um programa (Guia\_1606.txt) para o processador 8085 para calcular o quadrado de um dado de 8 bits mediante somas de ímpares.
- DICA: Somar os ímpares:  $5^2 = 1+3+5+7+9 = 25$ .

$$\text{dado02} = \text{SQR}(\text{dado01})$$

- 07.) Implementar um programa (Guia\_1607.txt) para o processador 8085 para calcular o fatorial de um dado de 8 bits. Usar funções.

$$\text{dado02} = \text{FAT}(\text{dado01})$$