1.a) Mapa de Karnaugh e simplificação da função F(A,B,C,D)

A função dada é:

$F(A,B,C,D)=\sum m(0,4,9,12,15)+\sum md(2,5,10,13)$

Passos:

1. **Construir o mapa de Karnaugh:**

Primeiro, cria-se o mapa de Karnaugh de 4 variáveis (A, B, C, D). Como A é a variável de maior peso, seguimos a ordem de A, B nas linhas e C, D nas colunas.

Mapa de Karnaugh (4 variáveis):

AB\CD	**00**	**01**	**11**	**10**
00	m(0)	m(1)	m(3)	m(2)
01	m(4)	m(5)	m(7)	m(6)
11	m(12)	m(13)	m(15)	m(14)
10	m(8)	m(9)	m(11)	m(10)

Coloca-se a informação fornecida:

- **Mintermos (1):** 0, 3, 9, 12, 15
- **Indiferenças (X):** 2, 5, 10, 13
- **Restante (0):** Todos os outros valores

O mapa fica:

AB\CD	**00**	**01**	**11**	**10**
00	1	0	0	X
01	1	X	0	0
11	1	X	1	0
10	0	1	0	X

2. **Simplificação (Soma de Produtos):**

A partir do mapa de Karnaugh, identificam-se os laços (grupos de 1s) para simplificar a expressão:

AB\CD	**00**	**01**	**11**	**10**
00	1	0	0	X
01	1	X	0	0
11	1	X	1	0
10	0	1	0	X

- **Grupo de 1s em m(0) e m(4):** /A /C /D
- **Grupo de 1s em m(4) e m(12) e X em m(5) e m(13):** B /C
- **Grupo de 1s em m(9) e X em m(13):** A /C D
- **Grupo de 1s em m(15) e X em m(13):** A B D

A expressão simplificada é:

F(A,B,C,D)= /A /C /D + B /C + A /C D + A B D

1.b) Simplificação da função para Produto de Somas

Para simplificar a expressão em produto de somas, duplica-se o mapa de Karnaugh e segue-se o processo de simplificação, laçando-se os 0´s. Observam-se os laços em termos de produtos de somas. A expressão simplificada em produto de somas através do Mapa de Karnaugh:

AB\CD	00	01	11	10
00	m(0)	m(1)	m(3)	m(2)
01	m(4)	m(5)	m(7)	m(6)
11	m(12)	m(13)	m(15)	m(14)
10	m(8)	m(9)	m(11)	m(10)

AB\CD	00	01	11	10
00	1	0	0	X
01	1	X	0	0
11	1	X	1	0
10	0	1	0	X

- **Grupo de 0s em m(8) e X em m(10):** /A + B + D
- **Grupo de 0s em m(1), m(3), m(7) e X em m(5):** A + /D
- **Grupo de 0s em m(6), m(14) e X em m(2) e m(10):** /C + D
- **Grupo de 0s em m(11) e X em m(10):** /A +B +/C

F (A,B,C,D) = **(/A + B + D) (A + /D) (/C + D) (/A +B +/C)**

2.a) Conversão de $7A4_h$ em base 7

O número é $7A4_h$, que está em hexadecimal. Converte-se esse número para base 7.

1. **Converter de hexadecimal para binário:**
   - 7 = 0111
   - A = 1010
   - 4 = 0100

   Assim, $7A4_h$ em binário é: **$0111\ 1010\ 0100_2$**

2. **Converter de binário para decimal (base 10):**

$(0x2^{11})+(1x2^{10})+(1x2^9)+(1x2^8)+(1x2^7)+(0x2^6)+(1x2^5)+(0x2^4)+(0x2^3)+(1x2^2)+(0x2^1)+(0x2^0)$

$=0+1024+512+256+128+0+32+0+0+4+0+0 = 1956$

3. **Converter de decimal (base 10) para base 7**

   Converter **1956_{10}** para **base 7** usando divisões sucessivas:

   Divisões:

   1956 ÷ 7 = 279, resto **3**

279 ÷ 7 = 39, resto **6**

39 ÷ 7 = 5, resto **4**

5 ÷ 7 = 0, resto **5**

Lendo os restos de baixo para cima = 5463_7

Logo, $7A4_h$ em base 7 é 5463_7

Em alternativa pode-se converte de hexadecimal para decimal e depois para base **septenária, sem necessidade de converter em binário.**

1. Hexadecimal para Decimal

$7A4_h = 7\times16^2 + 10\times16^1 + 4\times16^0$

$= 7\times256 + 10\times16 + 4$

$= 1792 + 160 + 4 = \mathbf{1956_{10}}$

Seguir o passo 3 anterior.

2.b) Conversão de 6547_8 para base 10

Para converter o número 6547_8 para base 10, utilizamos a fórmula de conversão de bases:

$$\mathbf{6547}_8 = 6 \times 8^3 + 5 \times 8^2 + 4 \times 8^1 + 7 \times 8^0$$

Calcula-se cada termo:

I. $6 \times 8^3 = 6 \times 512 = 3072$
II. $5 \times 8^2 = 5 \times 64 = 320$
III. $4 \times 8^1 = 4 \times 8 = 32$
IV. $7 \times 8^0 = 7 \times 1 = 7$

Somam-se os resultados:

3072 + 320 + 32 + 7 = 3431

O número 6547_8 em base 10 é 3431_{10}

3.a) Representação de -215 em binário com 12 bits (complemento de 2)

Para representar o número -215 em binário com 12 bits usando o complemento de 2:

1. **Representação de 215 em binário**: Converte-se o número positivo 215 para binário:

   $215 = 11010111_2$

   Como se usa 12 bits, adiciona-se quatro zeros à esquerda:

   $215 = 000011010111_2$

2. **Inverter os bits** (complemento de 1): Inverte-se todos os bits da representação binária:

   $000011010111_2 \rightarrow 111100101000_2$

3. **Somar 1 ao complemento de 1**: Adicionar 1 ao número obtido:

   $111100101000_2 + 1 = 111100101001_2$

A representação de -215 em binário com 12 bits (complemento de 2) é 111100101001_2

3.b) Represente o número $0{,}78125_{10}$ na base 20. Apresente os cálculos.

Para converter o número decimal 0,78125 para a base 20, seguimos um processo similar ao de conversão para outras bases.

Passo 1: Multiplicar a parte decimal por 20

A parte decimal do número é $0{,}78125_{10}$. Vamos multiplicar esse valor por 20 e observar a parte inteira e a parte decimal do resultado.

$0{,}78125 \times 20 = 15{,}625$

A parte inteira é 15, e a parte decimal é 0,625.

Passo 2: Obter o primeiro dígito na base 20

A parte inteira 15 corresponde ao primeiro dígito na base 20. Para representar esse número na base 20, usamos os seguintes símbolos:

- 0 a 9 representam os mesmos valores que na base 10.
- A representa 10,
- B representa 11,
- C representa 12,
- D representa 13,
- E representa 14,
- F representa 15,
- G representa 16,
- H representa 17,
- I representa 18.

Portanto, o primeiro dígito é F.

Passo 3: Continuar com a parte decimal

Com a parte decimal 0,625 repetimos o processo:

0,625 × 20 = 12,5

A parte inteira agora é 12, e a parte decimal é 5. O número 12 na base 20 é representado pela letra C.

Passo 4: Resultado final

Com a parte decimal 0,5 repetimos o processo:

0,5 × 20 = 10

O número 10 na base 20 é representado pela letra A.

Como a parte decimal agora é zero, paramos a conversão.

A representação de 0,78125 na base 20 é:

$0{,}78125_{10}=0{,}FCA_{20}$

Grupo II (5 valores)

Dada a função lógica: $F(A,B,C,D) = \left(A + \overline{B}C\right)\left(\overline{A} + BD\right) + \overline{\left(B\overline{C} + A\overline{D}\right)}$

4. Simplificação algébrica da função

Expandir o produto (A+/BC)(/A+BD)

Usar a distributiva:

=A/A + A·BD + /BC·/A + /BC·BD

Uma variável e seu complemento não podem ser verdadeiros ao mesmo tempo: A/A=0

=0 + A BD + /BC /A + /BC BD

Na última parcela, /B·B=0, logo:

/BC BD=0

Portanto:

=ABD + /BC /A

Passo 2: Analisar o segundo termo:

/(B/C+A/D)

Aplicando a Lei De Morgan:

=/(B/C)·/(A/D) = (/B + C) (/A + D)

Passo 3: A função fica:

F = ABD + /BC /A + (/B + C) (/A + D)

Passo 4: Expandir (/B + C) (/A + D)

=/B/ A + /BD + C/A + CD

Passo 5: Substituir e reescrever F:

F = ABD + /BC /A + /B/ A + /BD + C/A + CD

Passo 6: Agrupar termos semelhantes para simplificar

Notar que BC /A está contido em C/A logo:

BC /A + C/A = C/A considerando que C/A + BC /A = C/A (1 + /B) = C/A

Passo 7:

F = ABD + C/A + /B/A + /BD + CD

Passo 8: Procurar termos que podem ser combinados

1. ABD e /BD:

   Podemos escrever D(AB + /B)

Note que:

AB + /B = /B + AB

Utilizando o teorema do complemento:

/B + AB = (/B + A)(/B + B) = (/B + A)·1= /B + A

Portanto:

D(AB + /B) = D(/B + A)

Passo 10: Substituir em F:

F = C/A + /B/A + D(/B + A) + CD

Passo 11: Agrupar C/A + /B/A:

/A (C+/B)

Passo 12: Escrever F final simplificada:

F=/A(C + /B)+D(/B+ A) + CD

5 Implementação com Portas NAND

F=**/A(C + /B)**+**D(/B+ A)** + **CD**

Equivalências:

NOT(A) = /A=NAND(A,A)

AND(A, B) = A·B = / NAND(A,B) = NAND(NAND(A,B),NAND(A,B))

OR(A, B) = A+B = /(/A·B) =NAND(NAND(A,A),NAND(B,B))

Termo 1: **/A(C + /B)**

/A = NAND(A,A)

/B = NAND(B,B)

C+ /B =NAND(NAND(C,C),NAND(B,B))

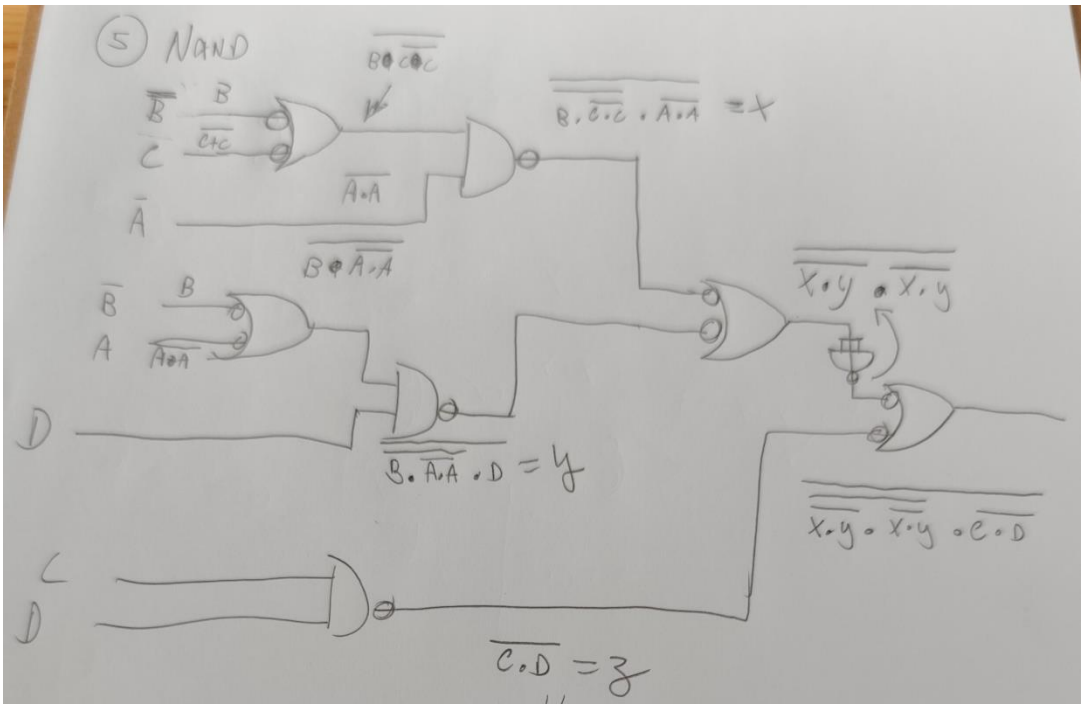
Termo 2: **D(/B+ A)**

/B = NAND(B,B)

/B +A=NAND(NAND(B,B),NAND(A,A))

Termo 3: **CD**

C D = NAND(C,D)



6. Implementação com Portas NOR

NOR(X,Y)= /(X+Y) , a porta NOR é a negação da porta OR

/X = NOR(X,X) , NOT usando NOR

X+Y= / (NOR(X,Y)) = NOT(NOR(X,Y)) = NOR(NOR(X,Y),NOR(X,Y)) , OR usando NOR

X·Y= /(/X + /Y) = NOR(/X,/Y) = NOR(NOR(X,X),NOR(Y,Y)) , AND usando NOR (lei de De Morgan)

F= **/A(C + /B)+D(/B+ A)** + **CD**

Termo 1: **/A (C + /B)**

/A = NOR(A,A)

/B = NOR(B,B)

C + /B = OR(C, /B) = NOR(NOR(C, /B),NOR(C, /B))

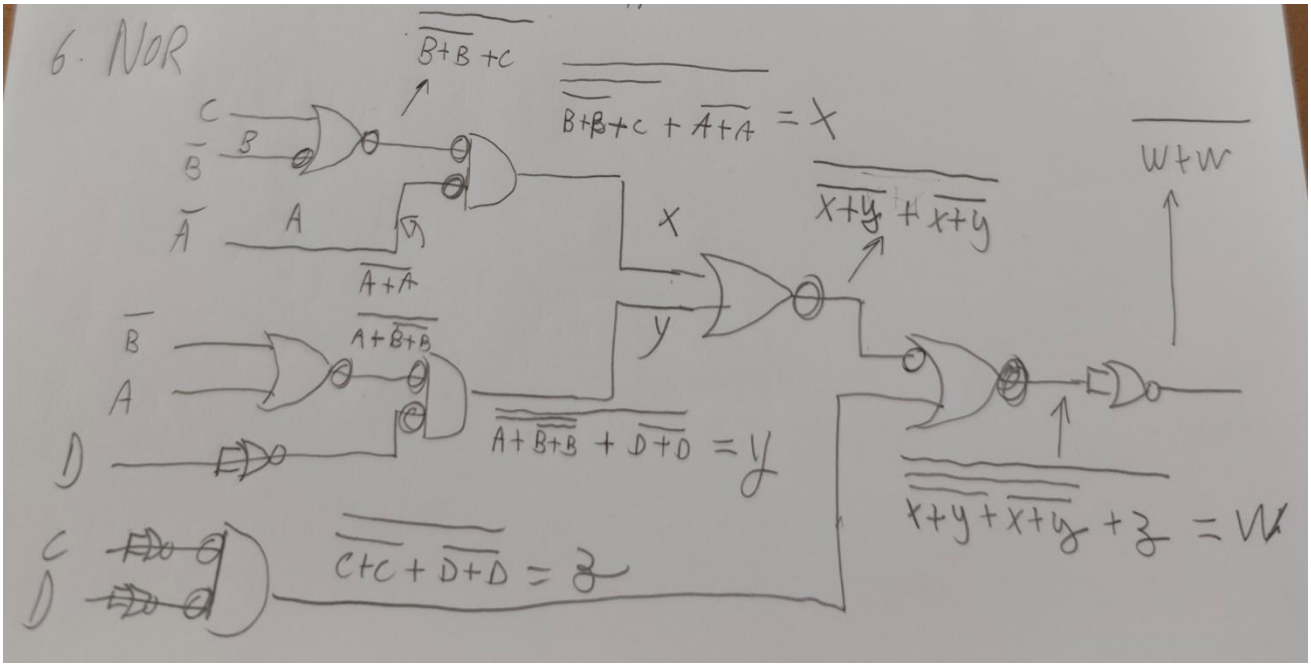
Termo 2: **D(/B+ A)**

/B = NOR(B,B)

/B +A = NOR(NOR(/B ,A),NOR(/B ,A))

Termo 3: **CD**

CD = NOR(NOR(C,C),NOR(D,D))



7. Implementação com Multiplexer

Para implementar F(A, B, C, D) usando um **multiplexer de 2 variáveis de seleção**:

1. **Escolha das variáveis de seleção:**
   - S_2 = A, S_1 = B

2. **Tabela verdade para F(A, B, C):**

A	B	F(C)
0	0	(0 + /0C)(/0 + 0D)+ /(0/C + 0/D)= (1) (1) + /(0+0) = 1+1 = 1
0	1	(0 + /1C)+(/0+ 1D)+/(1/C + 0/D)= (0+0C)(1+D)+/(/C +)= (0)(1) + //C = 0+ C= C
1	0	(1 + /0C)+(/1+ 0D)+/(0/C + 1/D)= (1)(0)+/(0+/D)= 0+D= D
1	1	(1 + /1C)+(/1+ 1D)+/(1/C + 1/D)= (1+0)(0+D)+/(/C+/D)= 1 D + /(/C+/D)= D + CD= D(1+C)= D

$F(A,B,C,D)$ = $(A+/BC)(/A+BD)$+/(B/C+A/D)

3. **Entradas do Multiplexer (I_0, I_1, I_2, I_3):**

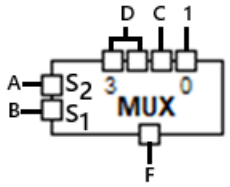
- I_0 = 1 (caso A=0, B=0)
- I_1 = C (caso A=0, B=1)
- I_2 = D (caso A=1, B=0)
- I_3 = D (caso A=1, B=1)

4. **Configuração do Multiplexer:**

   Conectar A e B às entradas de seleção S_2 e S_1, respetivamente, e conectar os valores I_0, I_1, I_2, I_3 às entradas de dados do multiplexador.

Expressão final baseada no multiplexer:

F(A,B,C) = MUX (A,B, I0=1, I1=C, I2=D, I3=D)



8. Diagrama de Estados

Máquina de Mealy com 4 Estados para Deteção de "1101"

1. **Definição dos estados**:
   - S0: Estado inicial (nenhum bit da sequência).Primeiro 1 detetado
   - S1: 1 detetado
   - S2: 0 detetado
   - S3: Sequência "1101" detetada

2. **Transições de estados**:
   - De S0:
     - **Entrada 1=0: Permanece em S0**
     - **Entrada 1=1: Transita para S1**
   - De S1:
     - **Entrada 1=0: Transita para S0**
     - **Entrada 1=1: Transita para S2**
   - De S2:
     - **Entrada 1=0: Transita para S3**
     - **Entrada 1=1: Permanece em S2 (múltiplos 1)**
   - De S3:
     - **Entrada 1=0: Transita para S0**
     - **Entrada 1=1: Transita para S1 com Saída Z=E2**

3. **Saída Z**:
   - Z = Entrada 2 somente no estado S3 para S1 com Entrada 1 = 1 (E1=1)
   - Z = 0 nos demais estados

Mapa de atribuição de estados

Um mapa de estados normalmente mostra, para cada combinação de entradas, qual será o próximo estado da máquina, com base no estado atual e nos valores de entrada.

Com 4 estados, precisamos de 2 bits para codificá-los (2^2 =4).

X0 \ X1	0	1
0	S0	S1
1	S2	S3

9. Tabela de transição de estados

	Atual		Entradas	Seguinte		Estado Seguinte	Saída
	X1	**X0**	**E1**	**X1**	**X0**		**Z**
S0	0	0	0	0	0	S0	0
S0	0	0	1	0	1	S1	0
S1	0	1	0	0	0	S0	0
S1	0	1	1	1	0	S2	0
S2	1	0	0	1	1	S3	0

S2	1	0	1	1	0	S2	0
S3	1	1	1	0	1	S1	Z= E2
S3	1	1	0	0	0	S0	0

10. Simplificação da variável de saída

Condição para Z = E2

- S3 = Q1 Q0 = 11
- E1=1

A saída Z depende de:

- Estado atual Q1=1, Q0=1
- Entrada E1 = 1
- A saída Z = E2 apenas nessa condição

Expressão booleana para Z:

- $Z=Q1 \cdot Q0 \cdot E1 \cdot E2$

Q1 e Q0 são os bits do estado atual.

Z=E2 apenas quando a Máquina de Estados Finitos está em S3 (Q1=1,Q0=1) e E1=1.

Caso contrário, Z=0.

11.a) [0,5] Coloca em R1 os seus bits deslocados uma unidade para a esquerda, sendo mantido o bit mais significativo

SHLA R1, 1

SHLA
Formato: SHLA op, const *Flags*: ZCNO
Acão: *shift left arithmetic*, mesma operação que SHL , mas atualizando os bits de estado correspondentes às operações aritméticas. Permite realizar de forma expedita uma multiplicação de op por 2n. O valor de const tem que estar compreendido entre 1 e 16.

Em alternativa

SHL R1, 1

SHL

Formato: SHL op, const *Flags*: ZCN

Acção: *shift left*, deslocamento à esquerda dos bits de op o número de vezes indicado por const . Os bits mais à esquerda de op são perdidos e é colocado 0 nas posições mais à direita. O bit de estado transporte fica com o valor do último bit perdido. O valor de const tem que estar compreendido entre 1 e 16.

11.b) [0,5] Colocar na posição de memória em R1 o conteúdo de R2.

MOV [R1], R2

MOV
Formato: MOV op1, op2 Flags: Nenhuma
Acão: op1 op2, copia o conteúdo de op2 para op1.
Para além dos modos de endereçamento comuns a todas as instruções (conforme Secção 3.4), esta instrução permite ler e escrever no registo apontador da pilha SP, mas apenas em conjunção com o modo de endereçamento por registo: MOV SP, Rx e MOV Rx, SP. A primeira destas instruções será necessária no início de todos os programas que utilizem a pilha.

11.c) [0,5] Chamada condicional à subrotina "rotina", se a última operação aritmética/lógica teve resultado zero

CALL.Z rotina ; melhor opção

CALL.NP rotina ; Resultado não positivo (≤ 0)
CALL.NN rotina ; Resultado não negativo (≥ 0)

CALL.*cond*

Formato: CALL.*cond* <endereço> *Flags*: Nenhuma

Acção: chamada condicional a uma subrotina baseado no valor de um dado bit de estado. As versões disponíveis são:

Condição	Transporte	Sinal	Excesso	Zero	Interrupção	Positivo
Verdade	CALL.C	CALL.N	CALL.O	CALL.Z	CALL.I	CALL.P
Falso	CALL.NC	CALL.NN	CALL.NO	CALL.NZ	CALL.NI	CALL.NP

Caso a condição se verifique, comporta-se como uma instrução CALL. Caso contrário, funciona como um NOP. Normalmente <endereço> é especificado com uma etiqueta.

1. d) [0,5] Colocar na pilha o valor da constante "W"

MOV R4, W
PUSH R4

PUSH

Formato: `PUSH op` *Flags*: Nenhuma

Acção: *M[SP]* ← *op*, *SP* ← *SP - 1*, coloca `op` no topo da pilha.

O P3 não permite o modo imediato (constante direta) com a instrução PUSH, por isso é necessário primeiro carregar a constante num registo.

12. [3] Elabore um programa no assembly do P3. O programa recebe um vetor de números e transforma-o num vetor de diferenças consecutivas, retornando também o número de elementos válidos.

```
; --- Definição dos dados iniciais ---

    ORIG 1000h       ; Área de dados a partir do endereço 1000h

Vetor      WORD 10 ,20 , 30 ,25

; Posição 1000h: 10

; Posição 1001h: 20

; Posição 1002h: 30

; Posição 1003h: 25



; --- Programa principal ---

    ORIG 2000h       ; Código a partir do endereço 2000h

Inicio:     MOV R1, Vetor    ; R1 = endereço inicial do vetor (1000h)

    MOV R2, 4        ; R2 = número de elementos (4)



    ; --- Processamento ---

    MOV R3, R2       ; Copia tamanho original

    DEC R3           ; R3 = novo tamanho (3)

    MOV R4, R1       ; R4 = ponteiro atual (inicia em 1000h)

    MOV R5, R1       ; R5 = ponteiro próximo

    INC R5           ; R5 = 1001h (próxima posição)

Loop:     MOV R6, [R5]     ; Carrega vetor[i+1]

    SUB R6, [R4]     ; Calcula diferença

    MOV [R4], R6     ; Armazena no vetor[i]



    INC R4           ; Avança ponteiro atual

    INC R5           ; Avança ponteiro próximo
```

```
    DEC R2          ; Decrementa contador

    CMP R2, 1       ; Verifica se terminou

    BR.NZ Loop      ; Continua se não terminou

Fim:    HALT            ; Termina execução                      ; Fim do programa
```