

UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE
LABORATÓRIO DE SISTEMAS DIGITAIS

MONTADOR DO “PATINHO FEIO”

Antonio Marcos de Aguirra Massola
João José Neto
Moshe Bain

Julho
1977

Em memória de
Laís Costa Ortenzi

INDICE

<u>Assunto</u>	<u>Página</u>
CAPÍTULO 1 - INTRODUÇÃO	1
CAPÍTULO 2 - ARITMÉTICA BINÁRIA E HEXADECIMAL	1
Bases de Numeração	1
Bases mais empregadas em computação	2
Conversão entre as bases dois, dez e dezesseis	3
Soma de números binários positivos	6
Representação de números negativos	8
Aritmética no Patinho Feio	11
Blocos e Diagramas Lógicos	15
Exemplo de Programa Absoluto	18
Exemplos de Programas Relocáveis	23

1 - INTRODUÇÃO

O ante-projeto do minicomputador Patinho Feio nasceu de um curso de pós-graduação dado pelo Professor Glen George Langdon Jr., em 1972. A seguir, os engenheiros e estagiários do Laboratório de Sistemas Digitais (LSD) da EPUSP terminaram o projeto e montaram o Patinho Feio que, dessa forma, se tornou o primeiro computador projetado e construído no Brasil.

Os circuitos do Patinho Feio são totalmente constituídos por circuitos integrados da família TTL (“transistor - transistor logic”), apresentando uma memória de núcleos de ferrite, e tendo um ciclo de máquina de dois microsegundos.

O Patinho Feio foi destinado a pesquisas no LSD, tanto na área de programação (“software”) como dos circuitos eletrônicos (“hardware”).

Cuidou-se do desenvolvimento de um “software” que permitisse um uso mais eficiente do minicomputador, já que, de início só se podia programá-lo em linguagem de máquina, manualmente, através do seu painel. Em particular, foi definida uma linguagem de montador (“assembly language”), que associa a cada instrução de máquina um mnemônico, e um programa montador (“assembler”), cuja função é traduzir programas escritos em linguagem de montador para linguagem de máquina, os quais são os assuntos tratados neste manual.

Este manual foi escrito de forma a tratar cada tópico de forma mais ou menos extensa, na suposição de que o leitor tenha tido previamente apenas um pequeno contato com a área de computação, e pouco ou nenhum conhecimento de línguas de baixo nível, como um montador. Por causa disso, tentou-se fazer com que o manual fosse o mais auto-explicativo e independente possível de outros textos. Naturalmente é impossível

que um texto seja completamente independente de outros; por isso, recomenda-se consultar outros textos, tais como manuais de operação do Patinho Feio e de seus equipamentos periféricos (de entrada/saída), textos sobre números binários, etc.

Foi feito um bom esforço para apresentar os conceitos com clareza e para padronizar as notações, com o objetivo de tornar o manual realmente útil. Contudo, certamente muitas falhas subsistem, de forma que são bem recebidas quaisquer sugestões e críticas de modo a melhorar o manual em futuras edições.

Observações:

- a) As informações contidas neste manual são as melhores que se pôde obter na época em que o manual foi escrito (setembro de 1975). Contudo, devido ao constante desenvolvimento de novos projetos de "hardware" e "software" para o Patinho Feio, alguns detalhes podem ter sofrido alterações até a presente data.
- b) Os programas e trechos de programa existentes no manual foram aí colocados por estarem sintaticamente corretos, mas não representam necessariamente exemplos de boa técnica de programação.

2 - ARITMÉTICA BINÁRIA E HEXADECIMAL

(Com números inteiros)

1. Bases de Numeração

Utiliza-se, na vida diária, a base decimal de numeração para representar os números. Isto significa duas coisas:

- a) existem dez algarismos com os quais todos os números são representados (pois a base de numeração é dez), a saber: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- b) emprega-se uma notação posicional onde está subentendido que, quando um algarismo é deslocado de uma posição para a esquerda, seu valor é multiplicado por dez. Por exemplo:

$$295 = 2 \times 10^2 + 9 \times 10^1 + 5 \times 10^0$$

Generalizando, quando se escreve o número $N = d_n d_{n-1} \dots d_2 d_1 d_0$ (sem sinal), onde os d_i ($i = 0, 1, 2, \dots, n$) são os seus algarismos (ou dígitos), está-se querendo dizer que: $N = d_n \times 10^n + d_{n-1} \times 10^{n-1} + \dots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$.

Nada obriga a que se use apenas a base dez. Na verdade, qualquer base b (inteira) pode ser escolhida para representar um número. Para tanto, escolhem-se b símbolos distintos (os algarismos da base) que representam os números de zero a $(b - 1)$. Escrevendo-se agora $n + 1$ algarismos adjacentes $d_n d_{n-1} \dots d_1 d_0$ e subentendida a notação posicional descrita acima, tem-se o número N representado por essa notação:

$$N = d_n \cdot b^n + d_{n-1} b^{n-1} + \dots + d_1 b^1 + d_0 b^0$$

Inversamente, pode-se provar que cada número N tem uma única representação, numa dada base b , que satisfaz as condições mencionadas acima.

Exemplo: Escolhendo $b = 3$, têm-se três algarismos;

convencionalmente usa-se 0, 1, 2. Então tem-se:

$$(1202)_3 = 1 \times 3^3 + 2 \times 3^2 + 0 \times 3^1 + 2 \times 3^0 = (47)_{10}$$

Pode-se começar a perceber a importância do que foi dito acima quando se considera que os computadores modernos trabalham sempre, em última análise, com a base dois.

2. Bases mais empregadas em computação

Além da base dez, que é de uso geral, empregam-se comumente as seguintes bases:

- a) base dois (binária) - necessita dois algarismos distintos para representar os números zero e um. Por convenção utilizam-se os símbolos 0 e 1. Um algarismo binário é também chamado "bit" (do inglês "binary digit").

A base dois é extremamente importante pois, como já foi citado, os computadores só entendem sequências de zeros e uns, que são usadas tanto para representar as instruções dadas à máquina quanto números propriamente ditos.

- b) base oito (octal) - utiliza os algarismos de 0 a 7. Não será aqui tratada com mais detalhes porque não é utilizada no Pátinho Feio, embora o seja em vários outros computadores.

- c) base dezesseis (hexadecimal) - os dígitos hexadecimais são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F; usados para representar os números de zero a quinze.

Exemplo: $(AB)_{16} = 10 \times 16^1 + 11 \times 16^0 = (171)_{10}$

A correspondência entre os valores binários, decimais e hexadecimais é apresentada na tabela seguinte (note-se que são necessários quatro bits para representar todos os dígitos hexadecimais na base dois).

<u>Decimal</u>	<u>Hexadecimal</u>	<u>Binário</u>
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

3. Conversão de números entre as bases dois, dez e dezesseis

Conforme já se deve ter percebido, surge frequentemente a necessidade de converter números escritos em uma base para outra. Para isso existem algoritmos gerais, dos quais são apresentados abaixo alguns casos particulares:

- a) Conversão para a base dez de números escritos em outra base. Basta escrever o número na forma $d_n \cdot b^n + \dots + d_0$ e efetuar as operações indicadas.

Exemplos: 1º) $(101111100001)_2$ para a base 10

$$= 1 \cdot 2^{11} + 0 \cdot 2^{10} + \dots + 0 \cdot 2^1 + 1 = (3041)_{10}$$

2º) $(BE1)_{16}$ para a base 10

$$= 11 \times 16^2 + 14 \times 16 + 1 = (3041)_{10}$$

Uma forma conveniente de fazer isso é dada nos diagramas abaixo:

$(BE1)_{16}$ para a base 10

$$\begin{array}{rcccc} & & 11 & 14 & 1 \\ & & \downarrow & & \\ \text{base} & & & & \\ \text{original} = 16 \times & 11 & 190 & (3041)_{10} \end{array}$$

Método usado: $11 \times 16 + 14 = 190$

$$\begin{array}{c} \downarrow \\ 190 \times 16 + 1 = 3041 \end{array}$$

$(10110)_2$ para a base 10

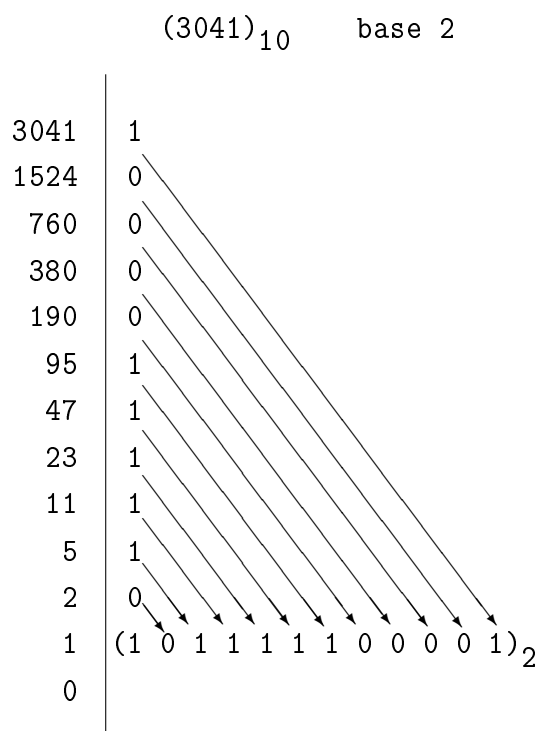
$$\begin{array}{rccccc} & & 1 & 0 & 1 & 1 & 0 \\ & & \downarrow & & & & \\ \text{base} & & & & & & \\ \text{original} = 2 & 1 & 2 & 5 & 11 & (22)_{10} \end{array}$$

Método usado: $1 \times 2 + 0 = 2$

$$\begin{array}{c} \downarrow \qquad \qquad \qquad \uparrow \\ 2 \times 2 + 1 = 5 \qquad \qquad 5 \times 2 + 1 = 11 \\ \qquad \qquad \qquad \downarrow \\ 11 \times 2 + 0 = 22 \end{array}$$

- b) Conversão de números escritos na base dez para uma outra base. Divide-se repetidamente o número dado pela base de destino até que o quociente seja zero. Os restos obtidos são a representação desejada, em ordem invertida. Ver os esquemas abaixo:

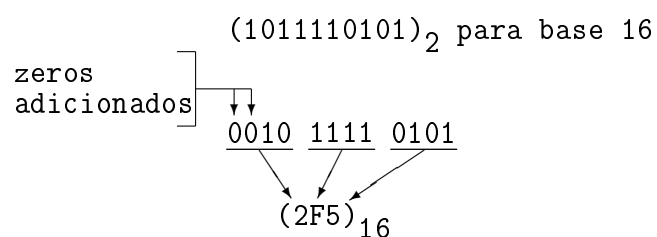
$$\begin{array}{rcl} (3041)_{10} & \text{base 16} & \\ \begin{array}{l} 3041 \\ 3041 : 16 \longrightarrow 190 \\ 190 : 16 \longrightarrow 11 \\ 11 : 16 \longrightarrow 0 \end{array} & \left| \begin{array}{l} \text{resto de } 3041 : 16 \\ \text{resto de } 190 : 16 \\ \text{resto de } 11 : 16 \\ \hline 1 \quad 14 \quad 11 \quad 14 \quad 1 \\ \quad \quad \quad \downarrow \downarrow \downarrow \downarrow \downarrow \\ \quad \quad \quad (BE1)_{16} \end{array} \right. & \end{array}$$



c) Conversão entre as bases dois e dezesseis.

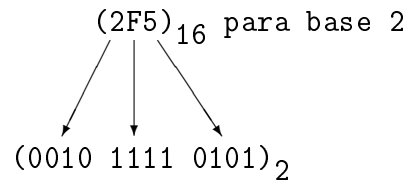
c-1) Da base dois para a base dezesseis. Basta agrupar os dígitos binários de quatro em quatro (a partir da direita) e substituí-los pelo respectivo dígito hexadecimal, conforme a tabela apresentada mais atrás (item 2. c).

Exemplo:



c-2) Da base dezesseis para a base dois. Basta substituir cada dígito hexadecimal pelo seu código de quatro bits.

Exemplo:



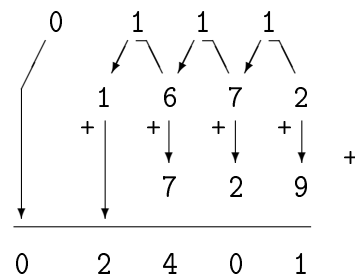
Obs.: 1º) Para a base oito, como é fácil perceber, o método é inteiramente análogo, dividindo-se o número binário em grupos de três bits.

2º) Pode-se agora notar porque são tão usadas as bases oito e dezesseis em computação: elas permitem dividir por três (pois $8 = 2^3$) e por quatro (pois $16 = 2^4$), respectivamente, o comprimento em algarismos do número escrito na base dois, que costuma ser inconveniente-mente longo.

3º) Estã-se dando mais ênfase à base hexadecimal porque no Patinho Feio os números têm ou oito ou doze bits de comprimento, podendo então, ser representados com dois ou três dígitos hexadecimais, enquanto que, por exemplo, para transformar um número de oito bits (também chamado “byte”) em um número octal, tem-se que adicionar um zero à frente do número, para dividi-lo em três grupos de três bits cada.

4. Soma de números binários positivos

Realiza-se de forma inteiramente análoga à soma comum, de números decimais. Para ver isso, examine-se detalhadamente uma soma decimal, por exemplo, de 1672 com 729.



Começando a partir da direita, foram realizadas as seguintes operações:

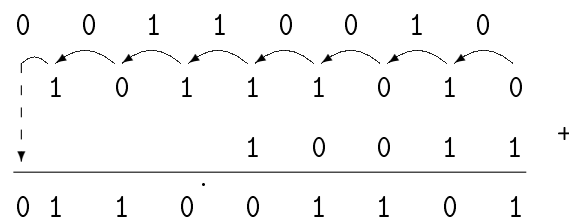
$$\begin{aligned}
 2 + 9 &= 1 \text{ e vai-um } (= 11) \\
 1 + 7 + 2 &= 0 \text{ e vai-um } (= 10) \\
 1 + 6 + 7 &= 4 \text{ e vai-um } (= 14) \\
 1 + 1 &= 2 \text{ e vai-zero } (= 02) \\
 0 &= 0
 \end{aligned}$$

Com os números binários procede-se da mesma forma, segundo as seguintes regras:

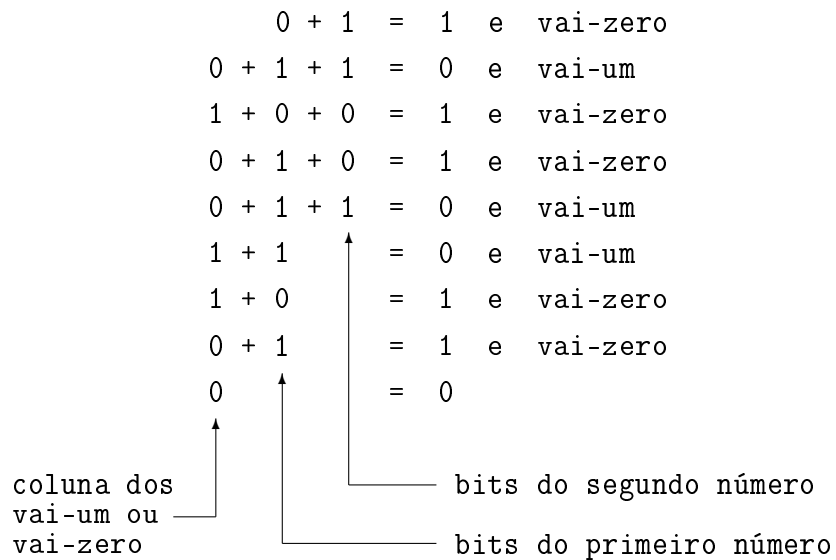
$$\begin{aligned}
 0 + 0 + 0 &= 00 \longrightarrow 0 \text{ e vai-um} \\
 0 + 0 + 1 &= 01 \longrightarrow 1 \text{ e vai-zero} \\
 0 + 1 + 1 &= 10 \longrightarrow 0 \text{ e vai-um} \\
 1 + 1 + 1 &= 11 \longrightarrow 1 \text{ e vai-um}
 \end{aligned}$$

Exemplo:

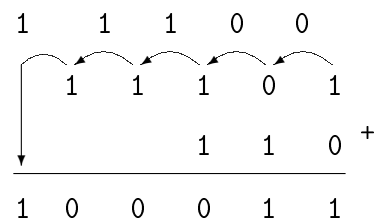
19) Seja somar 10111010 com 10011. Tem-se:



Começa-se a partir da direita, realizando as seguintes operações:



29) Somar 11101 com 110.



5. Representação de números negativos

Obs.: Nos itens seguintes assume-se sempre que um número tem oito bits de comprimento, quando for binário.

Até agora, só foram tratados os números positivos. Contudo, é óbvia a necessidade de se manipular números negativos, de modo que é preciso uma representação adequada para os mesmos. Especialmente, é necessária essa representação para números binários, de modo que o computador possa reconhecer os números que sejam negativos como tais.

Existem três modos de representar números negativos em notação binária, chamados de: sinal e amplitude, complemento de um e complemento de dois.

a) Representação de sinal e amplitude.

Usualmente, quando se quer denotar um número como negativo (em qualquer base), coloca-se à sua frente um sinal de menos (-), e quando positivo, às vezes, o sinal de mais (+). Mas, como um computador não reconhece os sinais + e -, mas apenas zeros e uns, vê-se que é necessário reservar um bit do número (geralmente o primeiro) para indicar o seu sinal (usa-se zero para indicar um número positivo e um para indicar um negativo). Supondo um número de oito bits, tem-se, por exemplo:

$(+6)_{10}$	0000 0110
$(-6)_{10}$	1000 0100
	└┐
	sinal amplitude

Desta forma pode-se representar os números inteiros de -127 a +127. Note-se que existem duas representações do número zero, a saber: 0000 0000 e 1000 0000.

b) Representação em complemento de um.

Nesta representação, para indicar um número negativo troca-se os seus zeros por uns e vice-versa. Como sempre, o primeiro bit indicará o sinal do número. Exemplo:

$(+6)_{10}$	0000 0110
$(-6)_{10}$	1111 1001
	sinal

Deste modo, analogamente ao anterior, pode-se representar os números de -127 a +127 e o zero continua com duas representações, a saber: 0000 0000 e 1111 1111.

c) Representação em complemento de dois.

Para se obter a representação em complemento de dois, soma-se um (em binário) à representação em complemento de um, re_tendo-se apenas os oito bits mais à direita. Exemplo:

$$\begin{array}{rcll}
 (+6)_{10} = 0000\ 0110 & \xrightarrow{\text{ida}} & 1111\ 1001_1 & \\
 \uparrow & \text{complemento} & & \\
 \text{sinal} & \text{de um} & & \\
 0000\ 0101_1 & \xleftarrow{\text{volta}} & 1111\ 1010 & = (-6)_{10} \text{ em complemento de dois.} \\
 & & \uparrow & \\
 & & \text{sinal} &
 \end{array}$$

A representação em complemento de dois tem as seguintes propriedades:

- 1a.) O primeiro bit do número indica o seu sinal: positivo se zero e negativo se um.
- 2a.) São representáveis os números de -128, cuja representação é 1000 0000; a +127, cuja representação é 0111 1111. Desta forma, o número -128 não tem complemento de dois. De fato, tem-se:

$$\begin{array}{rcl}
 -128 \rightarrow 1000\ 0000 & \rightarrow & 0111\ 1111_1 \\
 & & \hline
 & & 1000\ 0000 \rightarrow +128?
 \end{array}$$

que está errado, pois é a representação de -128, não de +128.

- 3a.) O número zero tem apenas uma representação: 0000 0000. De fato, tem-se:

$$\begin{array}{rcl}
 0000\ 0000 & \rightarrow & 1111\ 1111_1 \\
 & & \hline
 \text{[desprezado]} & \rightarrow & 10000\ 0000 \rightarrow 0000\ 0000
 \end{array}$$

6. Aritmética no Patinho Feio

Já foi visto como somar números binários positivos e como representar números negativos em oito bits. Deste modo, pode-se passar à soma (e subtração) de números de oito bits, que é o que o Patinho Feio consegue fazer diretamente através de seus circuitos eletrônicos. Ver-se-á também, como operar com números de mais de oito bits e como reconhecer quando o resultado de uma soma não pode ser representado em oito bits (isto é, o número é menor que -128 ou maior que +127).

a) Vai-um:

Denomina-se vai-um de uma soma entre dois números de oito bits ao vai-um na última soma realizada (bit mais significativo), ou seja, ao que seria o nono bit da soma (se fossem considerados números de nove bits). Exemplo:

$$\begin{array}{r}
 \begin{array}{cccccccc}
 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
 \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0
 \end{array} \\
 \begin{array}{cccccccc}
 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0^+
 \end{array} \\
 \hline
 \begin{array}{cccccccc}
 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array} \\
 \uparrow \\
 \text{vai-um}
 \end{array}$$

O vai-um funciona efetivamente como nono bit da soma, permitindo o tratamento de números de qualquer comprimento. Exemplo: seja somar os seguintes números:

$$\begin{array}{r}
 a = \quad 0101 \ 1101 \ 1010 \\
 + \\
 b = \quad 0110 \ 1001 \ 0110 \\
 \hline
 a+b = \quad 1100 \ 0111 \ 0000
 \end{array}$$

Dividindo a e b em duas partes de oito bits cada uma, vem:

$$\begin{array}{rcl}
 3^o) \quad -5 - 6 = -11 & 1111 \ 1011 & = -5_{10} \\
 & 1111 \ 1010 & = -6_{10} \\
 & \hline
 \text{vai-um}=1 & 1111 \ 0101 & = -11_{10}
 \end{array}$$

$$\begin{array}{rcl}
 4^o) \quad -5 + 5 = 0 & 1111 \ 1011 & = -5_{10} \\
 & 0000 \ 0101 & = +5_{10} \\
 & \hline
 \text{vai-um}=1 & 0000 \ 0000 & = 0
 \end{array}$$

Desta forma, consegue-se realizar qualquer soma e subtração, no Patinho Feio, de números de oito bits representados na notação complemento de dois, utilizando-se apenas a soma e a complementação de dois.

Obs.: Convém repetir que nada impede que se considere, se assim for conveniente, a sequência 1111 1010 como um número de oito bits desprovido de sinal (positivo) que valeria então $(250)_{10}$; analogamente 1111 1011 valeria $(251)_{10}$; somando-se estes dois números, obtém-se $(501)_{10}$, que em binário é 1 1111 0101; este é exatamente o resultado obtido na soma, desde que se considere o vai-um como nono bit da mesma.

c) Transbordo:

Como os números foram limitados a oito bits de comprimento, dos quais o primeiro é o bit de sinal, nota-se que existirão valores de a e b tais que, a+b ou a-b seja muito grande ou muito pequeno para ser representado em complemento de dois em oito bits. Contudo, é sempre possível fazer a soma tal como no item anterior, embora ela resulte errada. Neste caso diz-se que houve transbordo (do inglês “overflow”).

Exemplo:

$$(60)_{10} + (70)_{10} = (130)_{10} \text{ (o maior número representável é } +127)$$

$$\begin{array}{r} 0011 \ 1100 \\ + \ 0100 \ 0110 \\ \hline \text{vai-um}=0 \quad 1000 \ 0010 = (-126)_{10} \end{array} \text{ o que está obviamente errado}$$

Conclui-se, portanto, que é necessário ao computador detectar estes casos para evitar erros no processamento. Isto se faz comparando os dois últimos vai-uns ao realizar a soma: pode-se provar que, se eles diferirem entre si, houve transbordo. Exemplo: tomando o exemplo anterior, e detalhando todos os vai-uns, tem-se:

	0	1	1	1	1	1	0	0	
vai-um	0	0	1	1	1	1	0	0	= 60 ₁₀
+	0	1	0	0	0	1	1	0	= 70 ₁₀
	1	0	0	0	0	0	1	0	= -126 ₁₀

valores diferentes; indica
que houve transbordo
(resultado errado)

Tome-se agora o seguinte exemplo: $42 - 27 = 15$

	1	1	1	0	0	0	0	0	
vai-um	0	0	1	0	1	0	1	0	= 42 ₁₀
-	1	1	1	0	0	1	0	1	= -27 ₁₀
	0	0	0	0	1	1	1	1	

valores iguais; indica
que o resultado está
correcto

Pode-se, portanto, realizar qualquer soma e subtração de números de oito bits no Patinho Feio, obtendo-se não apenas o resultado em oito bits na mesma notação de complemento de dois usada nos operandos, como também um vai-um para funcionar como eventual nono bit da soma e uma indicação de transbordo que mostra se o resultado obtido está ou não correto.

7. Blocos e diagramas lógicos

Como serão apresentados mais adiante diagramas lógicos e funções lógicas executadas pelo Patinho Feio, tem-se abaixo, um pequeno resumo:

- a) Uma variável lógica é uma variável que pode assumir dois valores, geralmente chamados de 0(zero) e 1(um), ou F(falso) e V(verdadeiro), ou OFF(desligado) e ON(ligado).
- b) Uma função lógica associa a cada conjunto de valores de suas variáveis um dos dois valores citados (valor lógico).

Algumas funções são muito utilizadas e têm inclusive representação gráfica como um bloco lógico:

1º) Função NOT (negação) bloco:

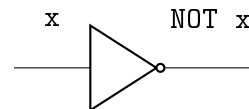
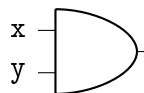


Tabela para a função NOT

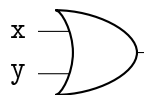
x	$x^1 = \text{NOT } x$
0	1
1	0

2º) Função AND (e) bloco:  x AND y

Tabela

x	y	$x \cdot y = x \text{ AND } y$
0	0	0
0	1	0
1	0	0
1	1	1

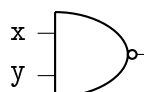
$x \cdot y$ só é um quando x e y forem ambos um.

3º) Função OR (ou) bloco:  x OR y

Tabela

x	y	$x + y = x \text{ OR } y$
0	0	0
0	1	1
1	0	1
1	1	1

$x + y$ vale um quando pelo menos uma das variáveis valer um.

4º) Função NAND bloco:  x NAND y
É um AND, seguido de um NOT.

Tabela

x	y	$(x \cdot y)^1 = x \text{ NAND } y$
0	0	1
0	1	1
1	0	1
1	1	0

$(x.y)^1$ vale um quando as duas variáveis não forem simultaneamente um.

g) Exemplo de Programa Absoluto:

```

MEM  E00
LER  E01
LEY  E7C
ARM  E34
SAI  E72
LIR  E23
LEX  E25
GUA  E31
PRE  E4F
VRC  E49
CAI  E4D
ENA  E5F
WAT  E65
WFF  E76
IGR  E7E
ACC  E84
LOP  E86
DIS  EC5
ACH  EB2
BRO  EB5
ARA  EB8

```

```
/00 SI
```

■PASS02

```

1  @BLTC
2  E00                ORG      /E00
3  *
4  *  HEXAM - PROGRAMA QUE CARREGA A MEMORIA
5  *          A PARTIR DE DADOS FORNECIDOS
6  *          EM HEXADECIMAL PELA CONSOLE
7  *
8  *****
9  *
10 *  HEXAM - INSTRUcoes DE UTILIZACAO:
11 *
12 *  1. ENDERECAR HEXAR
13 *  2. DAR PARTIDA
14 *  3. O CANAL B VAI FICAR ESPERANDO ENDERECAMENTO.
15 *  4. PARA ENDERECAR A QUALQUER MOMENTO, BATER ARROBA (@).
16 *  5. O COMPUTADOR RESPONDE C/ RETURN, 2 LINEFEEDS.
17 *  6. ENTRAR C/ ENDERECO EM HEXA, COM 3 DIGITOS
18 *  7. SE ERRAR, BASTA VOLTAR P/ 4 OU BATER UM BRANCO.
19 *      NESTE CASO, O PROGRAMA IGNORA A ENTRADA ANTERIOR
20 *      E AGUARDA NOVO ENDERECO.
21 *  8. UMA VEZ ENDERECADO, OS DADOS QUE FOREM FORNECIDOS
22 *      SERAO GUARDADOS EM SEQUENCIA A PARTIR DO ENDERECO
23 *      ESPECIFICADO.
24 *  9. OS DADOS DEVERAO VIR SEPARADOS POR UM UNICO BRANCO.

```

```

25 *10. O ULTIMO DADO DA LINHA NAO DEVE SER SEGUIDO DE BRANCO,
26 *   SENDO QUE NESTE CASO UM LINEFEED, RETURN OU VICE
27 *   VERSA O SUBSTITUIRA'.
28 *11. UM BRANCO OU RETURN DEPOIS DO DADO E' UMA ORDEM P/
29 *   QUE O DADO SEJA ARMAZENADO.
30 *12. DEPOIS DE CADA BRANCO OU RETURN O BUFFER E' ZERADO,
31 *   E PORTANTO SE FOREM DADOS 2 BRANCOS EM SEQUENCIA

```

```

32 *   SERA' GUARDADO UM ZERO NO LUGAR DO SEGUNDO BRANCO.
33 *13. EM CASO DE ERRO NOS DADOS, SE O CARACTER FORNECIDO
34 *   FOR HEXADECIMAL, BASTA BATER DE NOVO EM SEGUIDA, SEM
35 *   BRANCOS, O DADO CORRETO. SO' SAO GUARDADOS NA MEMORIA
36 *   OS DOIS ULTIMOS DIGITOS.
37 *14. SE O CARATER NAO FOR HEXADECIMAL, O COMPUTADOR RESPONDE
38 *   COM UMA SETA (_) E PARA O PROCESSAMENTO.
39 *15. NESTE CASO, DANDO PARTIDA, O PROGRAMA VOLTA A SER
40 *   EXECUTADO COMO NO CASO 14.
41 *16. ANTES DE DAR ENDERECAMENTO, E' PRECISO NAO ESQUECER
42 *   DE GUARDAR O DADO ANTERIOR. SE NAO FOR DADO UM BRANCO
43 *   OU RETURN, O DADO NAO SERA' ARMAZENADO.
44 *
45 *****
46 *
47 E00 9A      HEXAM   INIB      *          INIBE INTERRUPCAO
48 *
49 * SECAO DE LEITURA DE ENDERECO
50 *
51 E01 FE 7C   LEENDER PUG      LECONV  LE PRIMEIRO CAR. DO END.
52 E03 AE 01               PLAN    LEENDER SE BCO. OU RETURN, VOLTA
53 E05 D2 20               XOR     /20   NAO| MONTA "ARM"
54 E07 2E 34               ARM     ARM   GUARDA P/EXECUTAR
55 E09 FE 7C   PUG        LECONV  LE SEG. CARATER
56 E0B AE 01               PLAN    LEENDER SE BCO, VOLTA A LER ENDERECO
57 E0D D1 4F   DE         4       AJEITA P/ COMPOR
58 EOF 2E 35   ARM        ARM-1   GUARDA
59 E11 FE 7C   PUG        LECONV  LE TERCEIRO CARATER
60 E13 AE 01               PLAN    LEENDER SE BCO, VOLTA A LER END.
61 E15 6E 35   SOM        ARM+1   SE NAO COMPOE COM SEGUNDO DIG.
62 E17 2E 35   ARM        ARM+1   GUARDA P/ EXECUTAR
63 E19 DA 0D   CARI       /OD     SAI RETURN
64 E1B FE 72   PUG        SAI      NA TTY
65 E1D DA 0A   CARI       /OA     SAI LINEFEED
66 E1F FE 72   PUG        SAI      NA TTY
67 E21 FE 72   PUG        SAI      IDEM
68 *
69 * LEITURA DE UMA PALAVRA

```



```

70 *
71 E23 80      LIMPA  LIMPO  *      ZERA
72 E24 99      TRE    *      EXTENSAO
73 E25 FE 7C   LEPROX PUG    LECONV LE UM CARATER
74 E27 AE 31      PLAN  GUARDA SE BCO OU LINEFEED, STORE.
75 E29 99      TRE    *      SE NAO, TRAZ EXTENSAO
76 E2A 01 4F    DE     *      AJEITA PARA COMPOR
77 E2C 60 01    SOM    /001  COMPOE
78 E2E 99      TRE    *      C/ DIGITO LIDO
79 E2F 0E 25    PLA    LEPROX CONTINUA LENDO ATE' ACHAR BCO/RET
80 *
81 * ARMAZENAMENTO NA MEMORIA
82 *
83 E31 FE 4F   GUARDA PUG    PROTEGE TESTA SE ENDEREÇO INVADE
84 E33 99      TRE    *      HEXAM. SE NAO,
85 E34 20 00   ARM    ARM    *-*   GUARDA EXTENSAO NO ENDEREÇO CONV.
86 E36 4E 35      CAR    ARM+1 INCREMENTA
87 E38 85      INC     *      SEGUNDA PALAVRA
88 E39 2E 35    ARM    ARM+1 DO ENDEREÇO.
89 E3B 96      SV      1      SE NAO DEU CARRY,
90 E3C 0E 23    PLA    LIMPA  VAI LER A PROXIMA PALAVRA
91 E3E 4E 34    CAR    ARM    SE DEU,
92 E40 85      INC     *      INCREMENTA PRIMEIRA PALAVRA
93 E41 2E 34    ARM    ARM    GUARDA
94 E43 D1 0F    DD      4      TESTA SE DEU CARRY
95 E45 D8 FE    SOMI   -/02   ALEM DO BIT 11
96 E47 BE 23    PLAZ   LIMPA  NAO: VAI LER PROXIMA PAL.
97 E49 86      UNEG   UNEG    *      SIM: PARA EM LOOP
98 E4A 9D      PARE    *      COM /FF
99 E4B 0E 49    PLA    UNEG   NO ACUMULADOR
100 *
101 *
102 * PROTECAO DO PROGRAMA PARA QUE NAO SEJA DESTRUIDO PELOS DADOS
103 *      OU POR ENDERECAMENTO INVALIDO.
104 *
105 *
106 E4D DA 00   CARI    CARI    /00   RESTAURA ACUMULADOR
107 *
108 E4F 00 00   PROTEGE PLA    0      RETORNA
109 E51 2E 4E      ARM    CARI+1 SALVA ACUMULADOR
110 E53 4E 34      CAR    ARM    SEPARA 4 BITS
111 E55 D1 4F      DE     4      MAIS SIGNIFICATIVOS
112 E57 D1 6F      GE     4      DO ENDEREÇO
113 E59 D8 F2      SOMI   -/0E   TESTA SE ENDEREÇO >= /E00
114 E5B AE 4D      PLAN  CARI    NAO: VAI RETORNAR
115 E5D 0E 49      PLA    UNEG   SIM: VAI PARAR EM LOOP
116 *
117 * ENTRAS DRIVER DE ENTRADA DE DADOS, SEM BIT DE PARIDADE

```

```

118 *
119 E5F 00 00  ENTRA  PLA  0
120 E61 CB 11      FNC  /81  CLF
121 E63 CB 16      FNC  /86  STC
122 E65 CB 21  WAIT  SAL  /B1  ESPERA
123 E67 0E 65      PLA  WAIT  FLAG
124 E69 CB 40      ENTR  /B0  ENTRA DADO COMPLEMENTADO
125 E6B 82          CMP1  *    DESCOMPLEMENTA
126 E6C D1 41      DE    1    LIMPA PARIDADE
127 E6E D1 21      GD    1    DO DADO
128 E70 0E 5F      PLA  ENTRA  RETORNA
129 *
130 * SAI - DRIVER DE SAIDA
131 *
132 E72 00 00  SAI  PLA  0
133 E74 CB 80      SAI  /B0  SAI DADO
134 E76 CB 21  WFF  SAL  /B1  ESPERA
135 E78 0E 76      PLA  WFF  FLAG
136 E7A 0E 72      PLA  SAI  RETORNA
137 *
138 * LECONV = ROTINA DE "CONVERSAO"HEXBIN
139 *
140 E7C 00 00  LECONV  PLA  0
141 E7E DA 0F  IGNOR  CARI  /0F  FAZ INDICE
142 E80 9E      TRI  *    _/OF(NUMERO DE DIGITOS)
143 E81 FE 5F      PUG  ENTRA  OBTEM DADO
144 E83 83      CMP2  *    TROCA SINAL
145 E84 2E C4      ARM  ACC  SALVA DADO COMPLEMENTADO
146 E86 4E C4  LOOP  CAR  ACC  CARREGA DADO COMPLEMENTADO
147 E88 BE 7E      PLAZ  IGNOR  SE FFRM, IGNORA-0
148 E8A 7E C5      SOMX  DIGITOS  TESTA SE E' DIGITO
149 E8C BE 82      PLAZ  ACH  SE FOR, => ACH
150 E8E E0 00      S?S  0    SE NAO, APONTAR PROXIMO
151 E90 0E 86      PLA  LOOP  E VAI TESTAR NOVAMENTE
152 E92 4E C4      CAR  ACC  SE NAO DIGITO, RECUPERA DADO
153 E94 D8 20      SOMI  @    TESTA SE E' BRANCO
154 E96 BE B5      PLAZ  BRANCO  SIM: => BRANCO
155 E98 4E C4      CAR  ACC  NAO=> CARREGA DADO
156 E9A D8 0D      SOMI  /0D  TESTA SE E' RETURN
157 E9C BE B5      PLAZ  BRANCO  SIM=> BRANCO
158 E9E 4E C4      CAI  ACC  NAO=> TESTA SE
159 EA0 DB 0A      SOMI  /0A  E' LINEFEED

```

160	EA2	BE	7E		PLAZ	IGNOR	SIM=> IGNORA
161	EA4	4E	C4		CAR	ACC	NAO=> TESTA SE
162	EA6	D8	40		SOMI	@@	ARROBA
163	EA8	BE	B8		PLAZ	ARROBA	SIM=> ARROBA
164	EAA	DA	3F		CARI	@_	NAO=> INVALIDO!
165	EAC	FE	72		PUG	SAI	IMPRIME "_"NA CONSOLE
166	EAE	80			LIMPO	*	ZERA ACUMULADOR
167	EAF	9D			PARE	*	PARA
168	EB0	0E	7E		PLA	IGNOR	E IGNORA O CARATER
169	EB2	9E		ACH	TRI	*	(INDICE=DIGITO CONVERTIDO)
170	EB3	0E	7C		PLA	LECONV	JOGA NO ACC E VOLTA
171	EB5	86		BRANCO	UNEG	*	RETORNA C/ -1
172	EB6	0E	7C		PLA	LECONV	SE FOR BRANCO OU RETURN
173	*						
174	EB8	DA	0D	ARROBA	CARI	/0D	SAI
175	EBA	FE	72		PUG	SAI	RETURN
176	EBC	DA	0A		CARI	/0A	SAI
177	EBE	FE	72		PUG	SAI	DOIS
178	EC0	FE	72		PUG	SAI	LINEFEEDS
179	EC2	0E	01		PLA	LEENDER	VOLTA A LER ENDERECO.
180	*						
181	* BUFFERS	E	CONSTANTES				
182	*						
183	*						
184	EC4	00		ACC	DEFC	0	P/ SALVAR ACUMULADOR
185	EC5	30		DIGITOS	DEFC	@0	
186	EC6	31			DEFC	@1	
187	EC7	32			DEFC	@2	
188	EC8	33			DEFC	@3	
189	EC9	34			DEFC	@4	
190	ECA	35			DEFC	@5	
191	ECB	36			DEFC	@6	
192	ECC	37			DEFC	@7	
193	ECD	38			DEFC	@8	
194	ECE	39			DEFC	@9	
195	ECF	41			DEFC	@A	
196	ED0	42			DEFC	@B	
197	ED1	43			DEFC	@C	
198	ED2	44			DEFC	@D	
199	ED3	45			DEFC	@E	
200	ED4	46			DEFC	@F	

201 000

FIM

h) Exemplos de Programas Relocáveis:

```

CON 000 ENT
SUB *** EXT
ARF *** EXT
CAF *** EXT
SEN *** EXT
MAT 00E ABS
PIS 012
ACF 00A ABS
OFW 012 ABS

```

```

/00 SI

```

```

PASS02

```

```

1          @BLT
2 000          SUBR      COSEN
3          * ROTINA QUE CALCULA O COSENO NO PATINHO
4          * PELA FORMULA COS(X)= SEN(PI/2 - X)
5          *
6 000          EXT      COSEN
7 000          EXT      SUB
8 000          EXT      ARMACF
9 000          EXT      CARACF
10 000         EXT      SEN
11 000 00 00   COSEN    PLA      0
12 002 F0 00 X   PUG      ARMACF
13 004 01       DEFC      1
14 005 00 0E     DEFE     MANT
15 007 F0 00 X   PUG      CARACF
16 009 01       DEFC      1
17 00A 00 12 R   DEFE     PISDOIS
18 00C F0 00 X   PUG      SUB
19 00E F0 00 X   PUG      SEN
20 010 00 00 R   PLA      COSEN
21 00A          ACF      EQU     /00A
22 00E          MANT     EQU     /00E
23 012          OFLOW    EQU     /012
24 012 64       PISDOIS  DEFC     /64
25 013 87       DEFC     /87
26 014 D0       DEFC     /D0
27 015 01       DEFC     /01
28 000          PLA

```

DIY	000	ENT
SAA	***	EXT
NOM	***	EXT
NAM	***	EXT
TAB	***	EXT
ARF	***	EXT
SGL	***	EXT
COM	***	EXT
SOI	***	EXT
CAF	***	EXT
SHL	***	EXT
TAC	***	EXT
SHR	***	EXT
PON	***	EXT
RET	***	EXT
OFW	012	ABS
ZEO	017	ABS
FOO	01A	ABS
ACF	00A	ABS
MAT	00E	ABS
DFT	01E	ABS
GUI	016	
GOG	078	
SOS	05A	
MOE	030	
YES	045	
GOL	06E	

/00 SI

PASS02

```

1      @BLT
2 000      SUBR      DIV
3      *
4      *  DIV  -  ROTINA DE DIVISAO EM PONTO FLUTUANTE
5      *                      ACF = ACF/MANT
6      *
7 000      EXT      DIV
8 000      EXT      SALVA
9 000      EXT      NORM
10 000     EXT      NADABEM
11 000     EXT      TAB
12 000     EXT      ARMACF
13 000     EXT      SGNAL
14 000     EXT      COMPLEM
15 000     EXT      SOMATRI
16 000     EXT      CARACF
17 000     EXT      SHIFTL
18 000     EXT      TAC
19 000     EXT      SHIFTR
20 000     EXT      POESIN
21 000     EXT      REST
22 012     OFLOW    EQU      /012
23 017     ZERO     EQU      /017
24 01A     F        EQU      /01A
25 00A     ACF      EQU      /00A
26 00E     MANT     EQU      /00E
27 01E     DFLOAT   EQU      /01E
28      *
29 000 00 00  DIV    PLA    0
30 002 F0 00 X    PUG     SALVA  SALVA ACC,EXT,INDICE,T,V
31 TODO: Finish transcribing this...

```