UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE COMUNICAÇÕES

ISO / JBIG :

Codificação Aritmética de Imagens em 2 tons

Autor: Ahmed Mohamed Muftah Abushaala

Orientador: Prof. Dr. Yuzo Iano

Dissertação submetida à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos para a obtenção do Título de Mestre e Engenharia Elétrica.

Outubro 1996

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Ab97i

Abushaala, Ahmed Mohamed
ISO/JBIG: codificação aritmética de imagens em 2 tons / Ahmed Mohamed Abushaala.--Campinas, SP: [s.n.], 1996.

Orientador: Yuzo Iano.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Codificação. 2. Compressão de dados (Telecomunicações). 3. Processamento de imagens - Técnicas digitais. I. Iano, Yuzo. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Dedico este trabalho aos meus pais
e aos meus irmãos

يقول الرسول ﷺ :«مَنْ سلك طريقاً يلتمِسُ فيه علماً ، سهل الله به طريقاً إلى الجنة».

O Profeta Maomé *(mensageiro de Deus), que a paz esteja com ele, disse:*

"Àquele que persegue a senda do conhecimento,
Deus dirigirá para a senda do paraíso".

AGRADECIMENTOS

- Em primeiro lugar a Deus por tudo e sobre tudo;
- Aos meus pais e aos meus irmãos, pelo apoio, incentivo e inúmeras contribuições;
- Ao Prof. Dr. Yuzo Iano, pelo amizade, apoio e orientação;
- Aos amigos Abdalla Ibrahim Elusta, Almabruk Mansur Abogderah, Omar Mohamed Omar Gatuos, Ali Milad Almugla, Muftah Mohamed Basheer, Abdelhadi Ahmed Khalifa, Antônio Maria Fançony e Edson Adriano Vendrusculo pela colaboração e amizade;
- Aos colegas do curso de Mestrado e de Doutorado pela amizade;
- A todos os amigos do Departamento de Comunicações (DECOM), professores, alunos e funcionários;
- Às pessoas que, de alguma maneira, contribuíram para a realização deste trabalho;
- Por fim, gostaria de expressar minha profunda gratidão ao Centro de Pesquisa e Desenvolvimento da Telebrás (CPqD/Telebrás), ao Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), ao Fundação de Amparo a Pesquisa do Estado de São Paulo (FAPESP) e ao governo Brasileiro, pelo apoio financeiro e oportunidade oferecida que permitiram a concretização deste trabalho.

RESUMO

Neste trabalho abordou-se aspectos gerais sobre ***compressão de dados****, um tutorial sobre* ***codificação aritmética não binária*** *comparada com* ***codificação de Huffman*** *através de exemplos. A técnica de* ***codificação aritmética binária adaptativa*** *é estudada, tendo em vista sua aplicação em imagens de* ***2 tons*** *"bi-level images" tendo-se como base, técnicas de compressão padrão* ***ISO/JBIG*** *(****"International Standard Organization/Joint Bi-level Image experts Group"****). A imagem recuperada é igual à imagem original, ou seja, a técnica de compressão resulta em uma codificação sem perdas. Dedicamos nosso esforço para o estudo de* ***dois sub-blocos*** *que fazem partes do padrão JBIG: Sub-bloco* ***"Templates Modelos"*** *e sub-bloco* ***Codificador Aritmético Adaptativo*** *(CAA). O presente trabalho tem como objetivo, o estudo sobre codificação aritmética e desenvolvimento de* ***protótipo software do codec de codificação aritmética binária adaptativa para imagens em 2 tons.***

Palavras - Chave:

Codificação aritmética, codificação aritmética binária, codificação entrópica, padrão JBIG, codificação de imagem em dois tons ("bi-level"), compressão de dados.

CONTEÚDO

CAPÍTULO 5 : ***Codificação Aritmética Binário Adaptativa (ABAC - Adaptive Binary Arithmetic Coding)***

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS

JBIG : ***Joint Bi-level Image experts Group.***

ABAC : ***Adaptive Binary Arithmetic Coding.***

CCITT : ***The Consultative Committee of the International Telephone and Telegraph.***

D : ***Number of diferential layer.***

P : ***Number of plans..***

S : ***Number of stripe per layer.***

I_i : ***Image at layer i.***

X_i : ***Horizontal image size at layer i.***

Y_i : ***Vertical image size at layer i.***

R_i : ***Sample resolution.***

L_i : ***Lines per stripe at layer i.***

SEQ : ***Sequential.***

HITOLO : ***High to low.***

ILEAVE : ***Interleave multiple bit-planes.***

SMID : ***Index over stripe is in middle.***

$C_{s,d,p}$: ***Coded data for stripe s of resolution layer d of bit-plane p.***

LNT : ***Line not typical.***

VDP : ***Deterministic prediction value.***

VTP : ***Typical prediction value.***

MTA : ***Adaptive templates movement.***

MLNT : ***Same LNT.***

PCT : ***Stripe coded data.***

PIX : ***Pixel.***

CX : ***Context.***

LIFO : ***Last in first out.***

FIFO : ***First in first out.***

P_c : ***Cumulative probability.***

P : ***Relative probability.***

A , A_C : ***Current code interval.***

$A(.)$, A_N : ***New current code interval.***

C , C_C : ***Base of current code interval.***

$C(.)$, C_N : ***New bais of current code interval.***

FSM : ***Finite State Machine.***

BUF : ***External buffer.***

LPS : ***Less probable symbol.***

MPS : ***More probable symbol.***

LSZ : ***LPS size on coding interval.***

P_e : ***Current estimate.***

CT : ***Shift conter.***

SC : ***Stack conter.***

ST[cx] : ***State of CX.***

VMPS[cx]: ***Bit variable for more probabel symbol relative to CX.***

MPS/LPS : ***Conditional exchange.***

SWTCH : ***Switch.***

TEMP : ***Temproray variable.***

PBM : ***Portable BitMap.***

Capítulo 1

ASPECTOS GERAIS

1.1 Introdução

Com o avanço tecnológico da comunicação moderna, a demanda de transmissão e armazenamento de dados cresce rapidamente. Este crescimento requer eficiência e rapidez de execução dos algoritmos usados nas técnicas pertinentes à essa área, aproveitando-se as facilidades atuais existentes na forma de ferramentas e aplicativos disponíveis para computadores digitais.

Várias técnicas são usadas para transmitir ou armazenar grande quantidade de dados. Essas técnicas baseiam-se principalmente na compressão (*codificação*) de dados, as quais possibilitam que os mesmos sejam transmitidos em menor tempo e/ou armazenados no menor espaço possível. Algumas técnicas são implementadas utilizando-se uma imagem de 2 *tons* "***bi-level***" (isto é, qualquer imagem que possua duas cores, p. ex. uma imagem em preto e branco (***P & B***)).

Por razões que já foram citadas, existe uma grande quantidade de trabalhos que apresentam técnicas e métodos visando reduzir a quantidade de informação (**dados**) associados a um determinado sinal (*áudio ou vídeo*) a fim de se diminuir o volume do sinal (*tempo de transmissão, quantidade de memória física e largura de faixa*).

Entre os vários métodos existentes temos a ressaltar principalmente, a codificação de *Huffman* e a codificação aritmética.

Nos últimos anos, surgiu um padrão de compressão de dados denominado padrão ***JBIG*** (***"Joint Bi-level Image experts Group"***). Assim, decidiu-se para o presente trabalho utilizar um algoritmo de codificação aritmética binária adaptativa (*ABAC* - ***"Adaptive Binary Arithmetic Coding"***) para se implementar a técnica de compressão baseada no padrão *JBIG*. Utiliza-se para tanto uma imagem de 2 *tons*. Isso é discutido mais amplamente nos capítulos 2 e 5, onde se abordam diversos aspectos relacionados com a técnica escolhida.

No capítulo 3, discute-se a técnica de codificação de *Huffman* e a técnica de codificação aritmética (**não binária**) para extensão de alfabetos.

Como visto anteriormente, as duas técnicas de codificação são usadas para se reduzir a quantidade de informação necessária para transmissão e armazenamento de dados referentes a uma imagem. Essa redução é uma função da estatística da imagem.

1.2 A Compressão

O tempo de transmissão e a quantidade de armazenamento de informação de uma imagem são fatores muito importantes a serem levados em consideração na área de compressão de imagens. A importância desses fatores é evidenciada pelo grande interesse por parte de pesquisadores nesses dois temas. Portanto, surgiram vários métodos de compressão de informação de imagem abordando os dois aspectos. Uma classificação básica desses métodos de compressão se chama compressão física (codificação preditiva) [11],[12]. Nesse caso, explora-se a redundância presente na informação reduzindo-se a taxa de *bits* antes dos dígitos binários entrarem num meio de transmissão. A redundância é uma característica que está relacionada com fatores como predictibilidade, aleatoriedade, similaridade e outros fatores existentes na informação da imagem.

A partir do que foi frisado anteriormente, nós podemos definir o objetivo da compressão de informação de imagem como sendo:

A representação de informação de imagem por códigos mais eficientes sem redundância, de tal forma que a informação original possa ser recuperada a partir desses códigos, o que determina a taxa de informação, o tempo de transmissão e o espaço de memória de armazenamento.

Com relação a *transmissão de informação* [12], a compressão fornece diversos benefícios ao planejador de rede. Por exemplo, pode-se ter economia potencial de custo associada ao envio de menos dados. De fato, a redução do tempo de transmissão, pode reduzir a probabilidade de ocorrência de erros de transmissão porque há redução de informação. Aumenta-se assim a eficiência devido a tendência de se ter uma redução de jornadas extras de trabalho para o envio de informação completa. Além disso, fornece um nível maior de segurança pela conversão de texto para, por exemplo, o padrão ***ASCII***.

Com relação a *redução do volume de informação armazenada*, o tempo de execução do programa pode ser reduzido, podendo gerar uma redução no número de acessos ao disco.

Neste trabalho a **técnica de codificação sem perdas** é usada. Uma questão que está intimamente ligada à finalidade do serviço da técnica, seja para transmissão ou para armazenamento é a redução da taxa de transmissão bem como a capacidade de memória de armazenamento. Sendo assim, a eficiência da técnica de codificação é medida pela compressão (ou redução) do número de *bits* necessários para a transmissão, em comparação com o número de *bits* da imagem original.

Portanto, precisamos conhecer algumas terminologias que se utilizam para examinar ou avaliar a eficiência da técnica de compressão empregada. Temos :

- O grau de redução de dados obtidos pelo resultado do processo de compressão, é dado por:

$$\textit{razão de compressão} = \frac{\textit{quantidade de dados originais}}{\textit{quantidade de dados comprimidos}} \qquad (1.1)$$

De acordo com a eq. (1.1), quanto maior a razão de compressão, mais eficaz é a técnica de compressão empregada.

- Outro termo para exame é a figura de mérito que é dado por:

$$\textit{figura de mérito} = \frac{\textit{quantidade de dados comprimidos}}{\textit{quantidade de dados originais}} \qquad (1.2)$$

- O termo da fração da redução de dados é dado por:

$$\textit{fração da redução} = 1 - \textit{figura de mérito} \qquad (1.3)$$

- A porcentagem da compressão de dados é dada por:

$$\textit{\% compressão} = \textit{fração da redução} * 100 \qquad (1.4)$$

Assim, se uma dada técnica de compressão resulta em um símbolo de dado comprimido para cada 10 símbolos do fluxo de dados originais, ter-se-ia uma taxa de compressão de 10, uma razão de compressão de 10:1, uma figura de mérito de 0.1, uma fração da redução de dados de 0.9, e uma porcentagem de compressão de 90%.

Este trabalho é dedicado a uma técnica de codificação sem perdas da imagem digitalizada. A técnica de codificação aritmética binária adaptativa (*ABAC*) de padrão *JBIG* é usada para reduzir a redundância em transmissão de documentos por fac-símile digital em 2 *tons*. Esse ponto é melhor discutido no capítulos 2 e 5. O objetivo é preparar a imagem comprimida para transmissão ou armazenamento com o menor número possível de *bits*, preservando o nível da qualidade (requerida por aplicação específica relacionada com determinado serviço).

Vários tipos de imagem podem ser utilizados para essa técnica de compressão, por exemplo, imagens escaneadas de caracteres impressos, imagens de caracteres impressos geradas no computador, seis fontes de imagens típicas para fac-símile de

2 *tons* e um conjunto de 8 imagens para testes padronizadas pelo *CCITT* (**Comitê Consultivo Internacional de Telegrafia e Telefonia**). Sendo assim, cada amostra dessa imagem corresponde a um *pixel* que é representado por um **dígito binário** (***bit***).

1.3 Aplicações da Compressão em Comunicações

Sabendo-se que os custos de transmissão e de armazenamento de informação são fatores importantes em qualquer técnica de compressão, tais parâmetros são levados em consideração nas aplicações que requerem compressão de dados. Para *transmissão*, as técnicas de compressão são úteis para redução do tempo real requerido para enviar os dados, ou seja, a fim de reduzir a taxa de *bits* na linha de transmissão. Para *armazenamento*, as técnicas de compressão trabalham no sentido de reduzir a quantidade de dados que estão sendo armazenados.

Dessa forma, *as aplicações* para técnicas de compressão são diversas, por exemplo, transmissão de imagem de *TV* comercial via satélite, transmissão de dados via fac-símile, comunicações entre computadores, armazenamento de imagens médicas em sistemas de monitoramento de pacientes e assim por diante. De modo especial, várias técnicas são usadas para comprimir informações para transmissão por fac-símile. Um método eficiente é obtido ao se designar códigos pré-definidos para diferentes fileiras de *pixels* em preto e branco; os códigos substituem cada seqüência de fileiras de *pixels* encontrada em cada linha de varredura. Essa técnica é fundamentada na codificação de *Huffman* modificada [12]. Outro método mais eficiente ainda é obtido ao se designar códigos durante o processo de codificação. Esse método é utilizado na codificação aritmética [3].

1.4 Codificação sem Perdas

Sem recorrer à degradação da imagem, a codificação sem perdas (**codificação exata**) surge como opção real [7]. A recuperação dos dados deve ser efetuada sem

degradações acumulativas, isto é, a imagem recuperada é igual a transmitida ou armazenada. Em outras palavras, os dados recuperados são exatamente iguais aos dados da imagem original. A codificação usada para o fac-símile de documento - Grupo 3 do *CCITT* - é um exemplo prático de emprego da codificação sem perdas. Nessa aplicação, os vários comprimentos de correntes de **zeros e uns** têm palavras código cujos comprimentos respeitam suas probabilidades de ocorrência [14]. Quanto mais freqüentes, mais curtos. Esse método é chamado codificação com comprimento de palavra variável ("*Variable Length Code*"), ou algumas vezes chamado codificação entrópica.

Capítulo 2

PADRÃO *JBIG* (*Joint Bi-level Image experts Group*)

2.1 Introdução

O padrão *JBIG* foi preparado pelo grupo especialista em imagem 2 *tons* "*bi-level*", sendo que o grupo formou-se em 1988 para estabelecer um padrão para codificação progressiva de imagens em 2 *tons*.

A idéia do sistema de codificação progressiva é transmitir uma imagem comprimida, primeiro enviando-se os dados compactados em uma versão de resolução reduzida da imagem e depois enriquecendo-a o quando for necessário (ou requisitado) através da transmissão adicional de mais dados compactados, os quais complementam os dados já transmitidos anteriormente. A regulamentação ("***Recommendation | International Standard***") define um método de codificação como tendo os seguintes modos: *progressivo*, *seqüencial compatível-progressivo*, e *seqüencial progressivo-único*. Sugerem ainda uma maneira para se obter qualquer representação de baixa resolução que seja necessária.

Esse padrão define um método para codificação sem perdas de dados compactados de uma imagem em 2 *tons*. Isso se torna possível pela utilização da codificação citada e pelo uso de algoritmos de redução da resolução; isso para

codificação sem perdas de imagens em escala de cinza (“***grayscale***”) e imagens coloridas bem como para imagens de 2 *tons*.

A redução de resolução na **codificação** é realizada a partir das camadas de maior para as de menor resolução, enquanto a **decodificação** é realizada a partir das camadas de menor para as de maior resolução, isto é, a codificação divide a imagem original em várias ***camadas de resolução***.

A imagem de menor resolução é enviada numa seqüência progressiva sendo também uma imagem codificada seqüencialmente. Para uma aplicação de *codificação seqüencial progressiva-única*, somente essa imagem é enviada, sendo possível se criar um trem de pulsos no padrão *JBIG*. Na codificação sem modo progressivo, a imagem com resolução total é codificada sem referência a nenhuma camada de resolução ou seja, nesse caso, não se precisa dividir a imagem em camadas de resolução diferentes. Tal codificação é usada neste trabalho.

2.2 Os Benefícios da Codificação Progressiva

A codificação progressiva é do tipo adaptativo. Essa adaptação às características da imagem torna o método robusto em relação ao tipo da imagem. Em imagens escaneadas de caracteres impressos, as taxas de compressão observadas tem sido da ordem de 1,1 a 1,5 vezes maiores que aquelas obtidas pelo algoritmo de codificação *MMR* (*MR*: codificação de leitura modificada) descrito nas Recomendações T.4 (G3) e T.6 (G4). Em imagens de caracteres impressos geradas no computador, a taxa de compressão observada tem sido em torno de 5 vezes maiores. Em imagens com escala de cinza obtidas por “***halftoning***” ou “***dithering***”, as taxas de compressão observadas tem sido de 2 a 30 vezes maiores.

O método também tem capacidade progressiva. Quando uma imagem codificada progressivamente é decodificada, uma imagem de baixa resolução da imagem original é tornada disponível primeiro. Utilizam-se duplicações com as resoluções subsequentes aos dados já decodificados, à medida que mais dados vão sendo decodificados.

Usando-se codificação progressiva é possível se projetar uma aplicação com uma "*database*" comum a qual pode servir eficientemente os dispositivos de saída com capacidades de resolução extremamente diferentes. Então, somente aquela porção do arquivo da imagem compactada requerida para reconstrução de acordo com a capacidade de resolução do dispositivo de saída particular tem que ser enviada e decodificada. Também, se um enriquecimento adicional da resolução é desejado, por exemplo, no caso de uma cópia em papel de uma imagem já existente em uma tela de *TRC* (**Tubo de Raios Catódicos**), somente a informação extra de enriquecimento para a resolução necessária tem que ser enviada.

Outro benefício das codificações progressivas é que elas podem fornecer "*browsing*" em um *TRC* permitindo uma imagem de qualidade subjetiva superior para uma aplicação usando enlaces ("*links*") de comunicação de taxas-médias e taxas-baixas. Uma reprodução com baixa resolução é transmitida e mostrada rapidamente, e então seguida por outra de resolução maior com mais detalhes, tanto quanto forem desejados. Cada estágio extra de enriquecimento de resolução atua sobre a imagem já existente. A codificação progressiva possibilita ao usuário reconhecer rapidamente a imagem enquanto está sendo construída antes de se transmitir todas as informações correspondentes a imagem, e ainda permite ao usuário interromper o processo de construção da imagem se assim for desejado.

2.3 Estrutura Geral

O processo é realizado dividindo-se a imagem em várias partes, chamadas de camadas de resolução, antes de se usar a operação de compressão. Essas camadas podem ser representadas em planos de *bits*. Cada camada de resolução é dividida em faixas horizontais chamadas tiras e em determinado número de linhas L_i ($i = D,D-1, \ldots ,0$), sendo que as tiras na mesma camada de resolução i são iguais.

Seja D o número de duplicações em resolução (**camadas de resolução diferenciais**) fornecido pela codificação progressiva. Seja P o número de planos de *bits*, onde P depende de quantos *bits* por *pixel* são usados. Se a mensagem ou o *pixel* tem 8

bits, então existe a possibilidade de se ter 8 planos de *bits*, isto é, P é o número de *bits* por *pixel*. Seja S o número de tiras em cada camada de resolução, onde, S é igual para todas as camadas de resolução. I_D será referenciada como uma imagem da maior camada de resolução; então a sua dimensão horizontal será X_D ; sua dimensão vertical será Y_D ; sua resolução de amostragem será R_D (pontos-por-polegada); e o número de linhas em cada tira será L_D . A figura (2.1) mostra a decomposição quando se tem 3 camadas de resolução, 3 tiras por camada e somente 1 plano de *bits*.

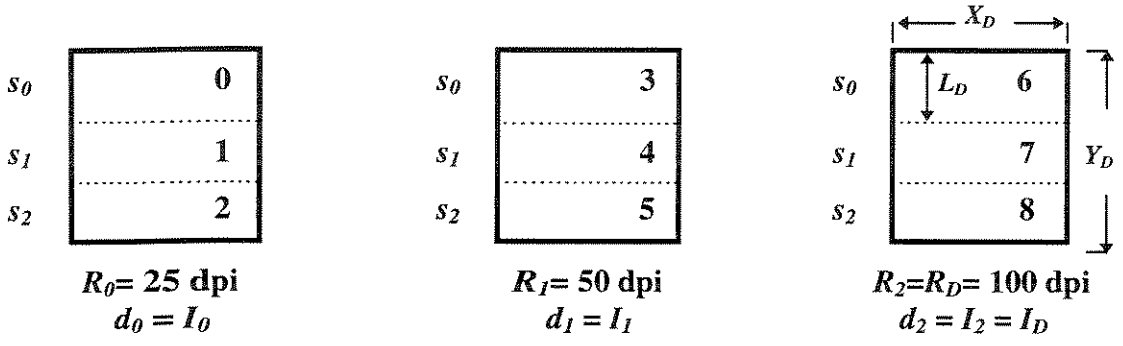


Fig. 2.1 - Decomposição fornecida pela codificação progressiva de : 3 camadas, 3 tiras e 1 plano de bits

A figura (2.2) mostra a decomposição quando se tem 3 camadas de resolução, 3 tiras por camada, e mais do que 1 plano de *bits*, digamos 2 planos de *bits*.

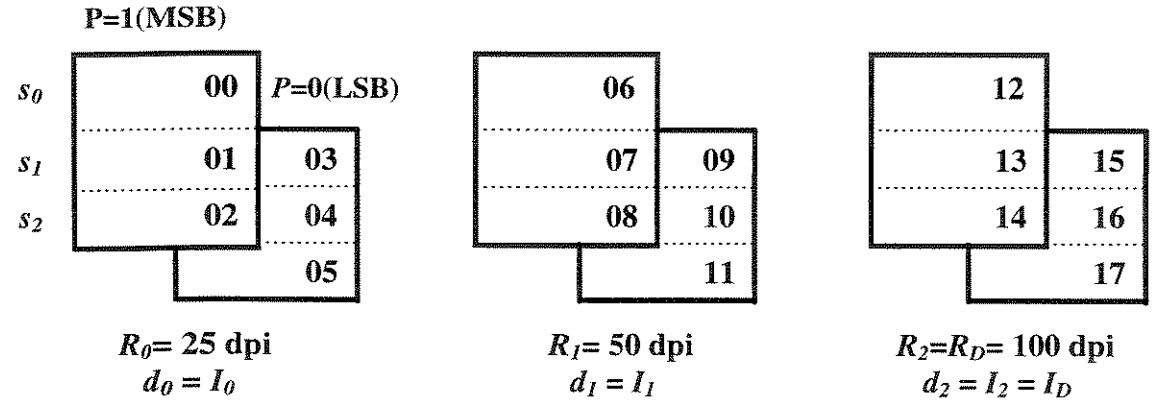


Fig. 2.2 - Decomposição fornecida pela codificação progressiva de : 3 camadas, 3 tiras e 2 planos de bits

A diferença entre a codificação progressiva e a forma tradicional de codificação da imagem está no fato de que a segunda codifica a imagem com sua resolução total enquanto que na primeira forma a resolução é distribuída pelas camadas. A varredura de resolução da segunda forma é feita da esquerda para a direita e de cima para baixo "**modo de *raster***" sem nenhuma referência às camadas de resolução, isto é, o número ***D*** de duplicações de resolução é igual a **zero**. Essa codificação é referida como sendo "**codificação seqüencial**".

A ordem do processo de codificação progressiva depende de algumas variáveis tanto para primeira forma que tem só 1 plano de *bits* quanto para a segunda forma que tem 2 planos de *bits*. A primeira forma depende de duas variáveis. Uma delas é a *SEQ bit* ("***Sequential***") ou seja, os dados são tratados pelo codificador do *JBIG* na forma seqüencial começando-se pela menor para a maior camada de resolução (ou vice versa) e pegando-se os dados da primeira tira seguidos pelos dados da segunda tira e assim por diante. Então se passa para a segunda camada de resolução repetindo-se o mesmo processo. Então nesse caso $SEQ = 0$. A outra variável é a *HITOLO bit* ("***High to low***") ou seja, o codificador do *JBIG* codifica os dados começando pela menor camada de resolução e depois indo para a maior camada de resolução ; então nesse caso $HITOLO = 0$. A tabela (2.1) mostra as possibilidades de ordem de figura (2.1).

Tab. 2.1 - As ordens possíveis nas nove tiras

HITOLO	*SEQ*	*Exemplos de ordem*
0	0	0, 1, 2, 3, 4, 5, 6, 7, 8
0	1	0, 3, 6, 1, 4, 7, 2, 5, 8
1	0	6, 7, 8, 3, 4, 5, 0, 1, 2
1	1	6, 3, 0, 7, 4, 1, 8, 5, 2

A segunda forma depende de mais duas variáveis, além das duas já citadas para a primeira forma; uma delas é a *ILEAVE bit* ("***Interleave multiple bit-planes***") ou seja, quando $ILEAVE = 1$, indica que o *bit* está ligado. A outra variável é a *SMID bit* ("***Index over stripe is in middle***") ou seja , quando $SMID = 1$, indica que S está no meio conforme

mostrado de maneira mais clara na tabela (2.2) a qual mostra também as possibilidades de ordem da figura (2.2).

Tab. 2.2 - As ordens possíveis no caso de multi-planos

HITOLO	*SEQ*	*ILEAVE*	*SMID*	*Exemplos de ordem*
0	0	0	0	(00,01,02 06,07,08 12,13,14) (03,04,05 09,10,11 15,16,17)
0	0	1	0	(00,01,02 03,04,05) (06,07,08 09,10,11) (12,13,14 15,16,17)
0	0	1	1	(00,03 01,04 02,05) (06,09 07,10 08,11) (12,15 13,16 14,17)
0	1	0	0	(00,06,12 03,09,15) (01,07,13 04,10,16) (02,08,14 05,11,17)
0	1	0	1	(00,06,12 01,07,13 02,08,14) (03,09,15 04,10,16 05,11,17)
0	1	1	0	(00,03 06,09 12,15) (01,04 07,10 13,16) (02,05 08,11 14,17)
1	0	0	0	(12,13,14 06,07,08 00,01,02) (15,16,17 09,10,11 03,04,05)
1	0	1	0	(12,13,14 15,16,17) (06,07,08 09,10,11) (00,01,02 03,04,05)
1	0	1	1	(12,15 13,16 14,17) (06,09 07,10 08,11) (00,03 01,04 02,05)
1	1	0	0	(12,06,00 15,09,03) (13,07,01 16,10,04) (14,08,02 17,11,05)
1	1	0	1	(12,06,00 13,07,01 14,08,02) (15,09,03 16,10,04 17,11,05)
1	1	1	0	(12,15 06,09 00,03) (13,16 07,10 01,04) (14,17 08,11 02,05)

Os dados compactados em "**palavras código**" $C_{s,d,p}$ para a tira s da camada de resolução d e do plano de *bits* p é independente da ordem de tiras. Devido à proposta de se tratar com apenas um plano de *bits*, então os dados compactados serão referenciados por $C_{s,d}$.

2.4 Blocos de Codificação

Um codificador pode ser decomposto em vários sub-blocos como mostrado na figura (2.3). Cada sub-bloco de redução de resolução e processamento de camada diferencial da figura (2.3) é idêntico aos demais sub-blocos em funções; por isso somente uma descrição da operação de uma camada é necessária. Para tal descrição, a imagem de chegada será referenciada como uma imagem de alta resolução (I_D) e a imagem de saída como uma de baixa resolução (I_{D-1}).

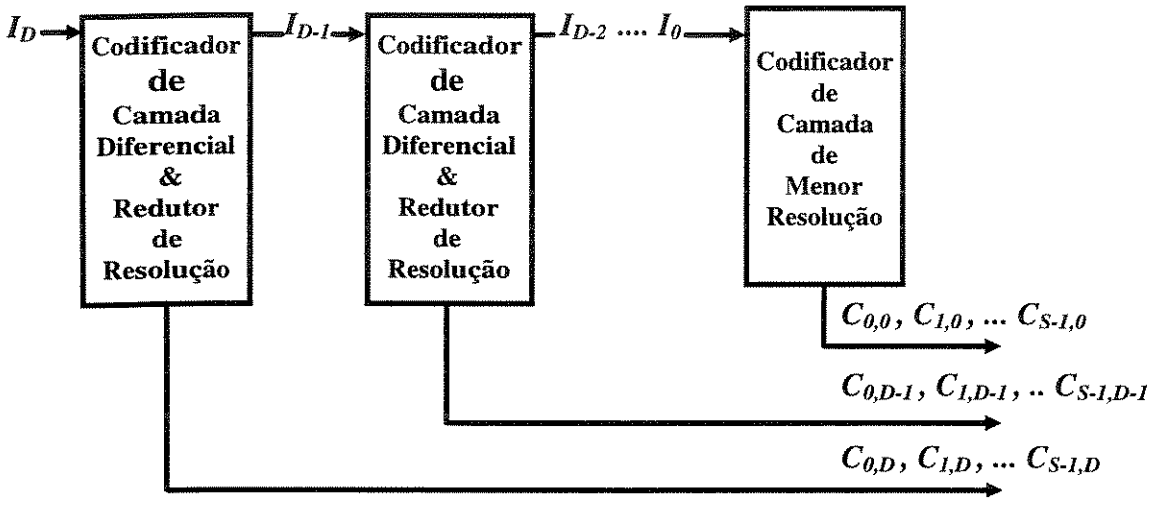


Fig. 2.3 - Sub-blocos do codificador

2.4.1 O Sub-bloco Codificador de Camada Diferencial e Redutor de Resolução

A figura (2.3) pode ser decomposta em sub-blocos como mostrado na figura (2.4) além do sub-bloco codificador de camada de menor resolução.

Tem-se :

$C_{s,d}$: Palavras código para tira s e camada d;

Is,d : Imagem na tira *s* e na camada *d*;

LNT : Linha não típica (sinal);

VDP : Valor determinístico de predição (sinal);

VTP : Valor típico de predição (sinal);

MTA : Movimento adaptativo da folha.

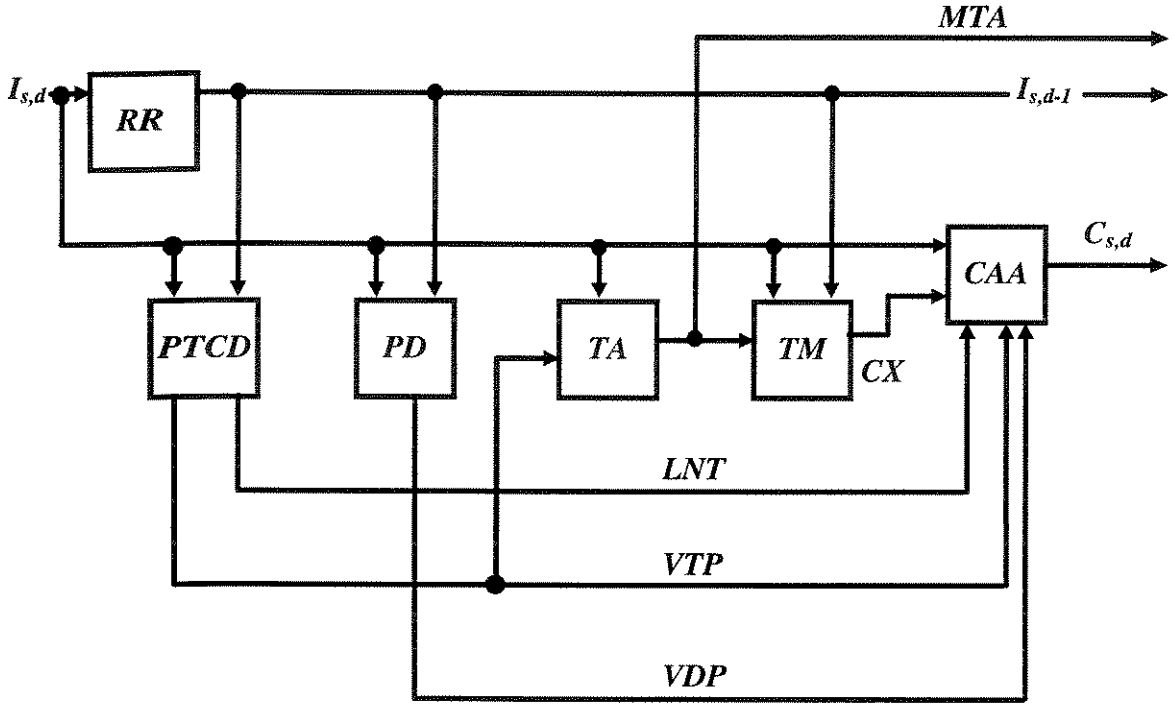


Fig. 2.4 - Codificador de camada diferencial e redutor de resolução

2.4.1.1 Redutor de Resolução (*RR*)

Esse bloco executa uma redução de resolução de tal forma que se aceita uma imagem de alta resolução e se cria uma de baixa resolução com qualidade tão próxima quanto possível da imagem correspondente a metade de todas as linhas e metade de todas as colunas da imagem de alta resolução.

2.4.1.2 Predição Típica de Camada Diferencial (*PTCD*)

Esse bloco procura por regiões de cor sólida (uniforme) e quando encontra um dado *pixel* corrente de alta resolução a ser codificado e que está em uma região como essa, nenhum dos processamentos normalmente feitos nos blocos *PD*, *TA*, *TM*, e *CAA* é necessário. Esse bloco proporciona alguns ganhos de codificação, também para acelerar as implementações. Em imagens de texto ou linhas desenhadas, o sub-bloco *PTCD* usualmente possibilita evitar a codificação de mais de 95% dos *pixels*.

2.4.1.3 Predição Determinística (*PD*)

O propósito desse bloco é fornecer ganho de codificação. Algumas vezes ocorre que o valor de um *pixel* de alta resolução a ser codificado é deduzido a partir dos *pixels* vizinhos particulares na imagem de baixa resolução e na imagem de alta resolução. Quando isso ocorre, o bloco *PD* sinaliza qualquer um de tais *pixels* e evita-se o processo de codificação no sub-bloco *CAA*. O sub-bloco *PD* é um algoritmo dirigido por tabelas *PD* ; essas tabelas são fortemente dependentes do método particular de redução de resolução usado.

2.4.1.4 "*Templates*" Adaptativos (*TA*)

O sub-bloco *TA* procura periodicidade na imagem sendo que encontrando uma periodicidade muda o "*template*". Quando isso ocorre, uma seqüência de controle *MTA* é adicionada ao fluxo de dados de saída ("*output datastream*"). Esse procedimento possibilita um substancial ganho de codificação (aproximadamente 80%) em imagens representadas em escala de cinza por "*halftoning*".

2.4.1.5 *"Templates"* Modelos (*TM*)

Para cada *pixel* de alta resolução a ser codificado, o sub-bloco *TM* fornece um número inteiro chamado contexto para o codificador aritmético. O codificador aritmético declara para cada contexto uma estimativa da probabilidade condicional do *pixel* dado àquele contexto.

2.4.1.6 Codificador Aritmético Adaptativo (*CAA*)

O sub-bloco *CAA* é um codificador de entropia. Ele observa as saídas dos sub-blocos *PTCD* e *PD* para determinar se é necessário codificar um dado *pixel*. Se for necessário, então o *CAA* observa o contexto e utiliza o seu estimador de probabilidade interno para estimar a probabilidade condicional do *pixel* corrente.

2.4.2 O Sub-bloco Codificador de Camada de Menor Resolução

Nesse caso, os sub-blocos *RR* e *PD* não são aplicáveis e os sub-blocos *PTCD*, *TA* e *TM* são diferentes porque não existe nenhuma camada de menor resolução para ser usada como entrada. A figura (2.5) mostra um codificador de camada de menor resolução.

Tem-se :

PTCF : Predição típica de camada funda, que tem mesma função de *PTCD*;

MLNT : Mesmo sinal *LNT*.

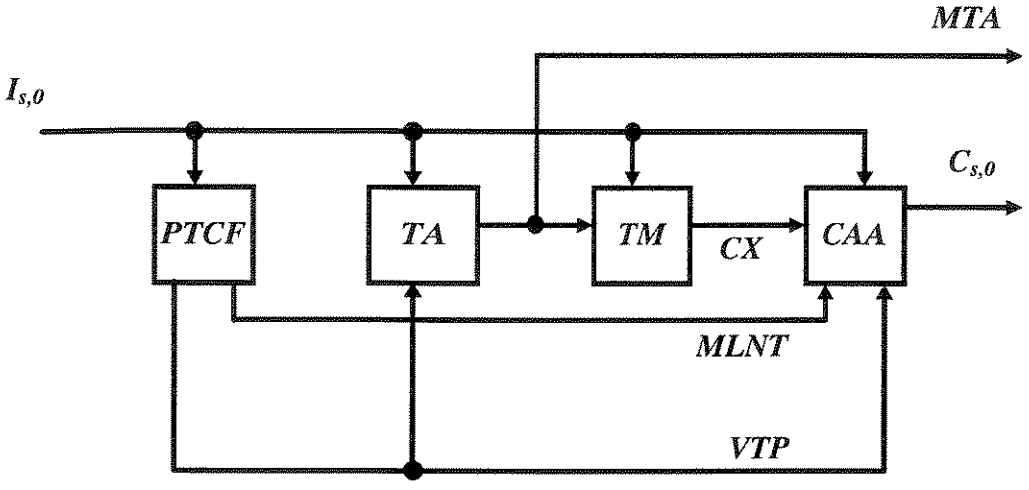


Fig. 2.5 - Codificador de camada de menor resolução

2.5 Blocos de Decodificação

Os sub-blocos do decodificador são análogos aos sub-blocos do codificador. Observe que os sub-blocos *TA* e *RR* não aparecem no decodificador. As figuras (2.6), (2.7) e (2.8) mostram a decomposição do decodificador para a camada diferencial de menor resolução.

Tem-se :

DAA : Decodificador Aritmético Adaptativo.

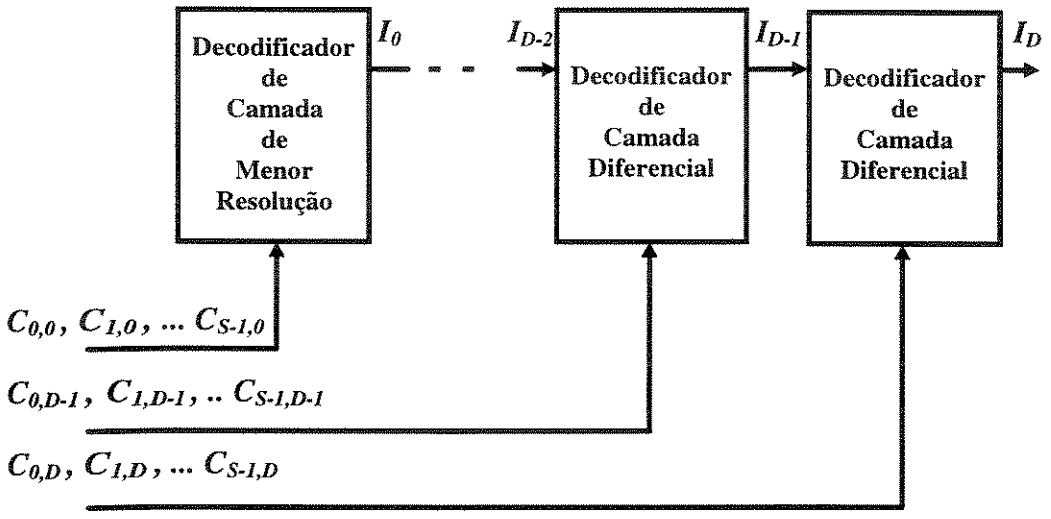


Fig. 2.6 - Sub-blocos do decodificador

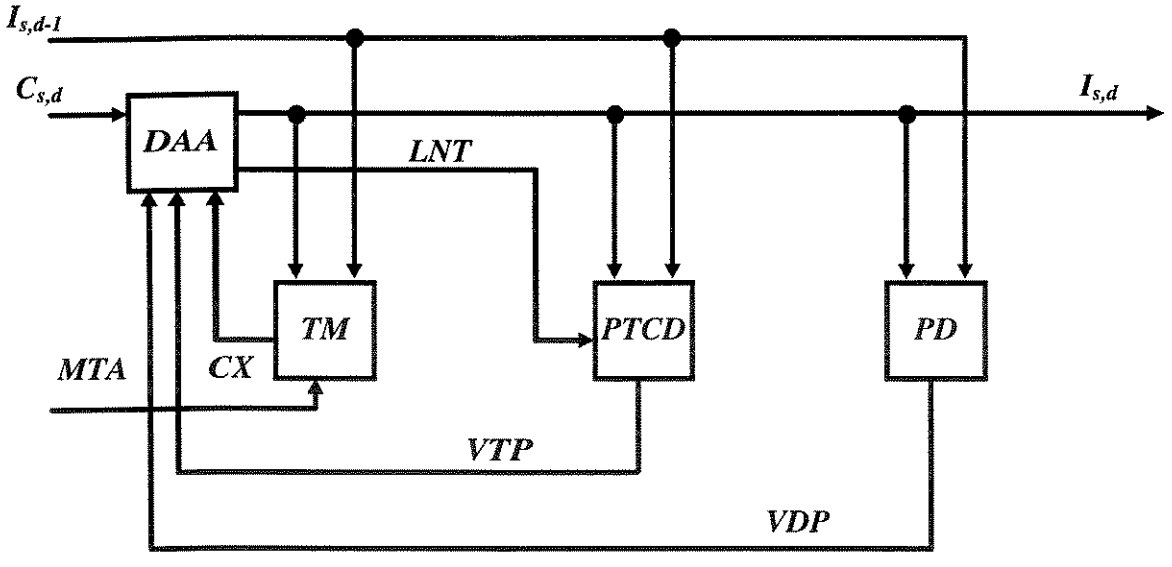


Fig. 2.7 - Decodificador de camada diferencial

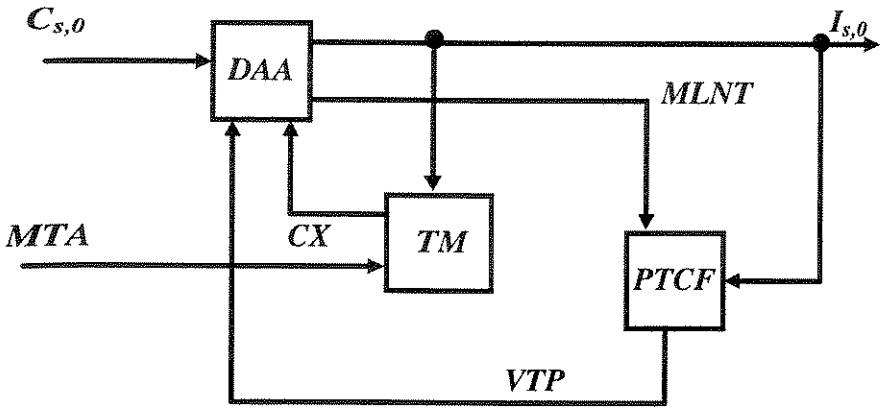


Fig. 2.8 - Decodificador de camada de menor resolução

Este trabalho enfoca dois sub-blocos dos seis existentes: sub-bloco de "*templates*" modelos e sub-bloco de aritmético adaptativo tanto para o codificador *CAA* como para o decodificador *DAA*.

Capítulo 3

CODIFICAÇÃO ESTATÍSTICA

3.1 Introdução

As codificações assim chamadas estatísticas são aquelas que utilizam técnicas de compressão baseadas na estatística da imagem. Isso significa que se codificam os dados com base na freqüência de ocorrência dos mesmos símbolos de dados. A codificação de *Huffman* e a codificação aritmética são consideradas como exemplo das diversas técnicas estatísticas de compressão de dados, que reduzem a redundância entre os dados e cujo emprego minimiza o tamanho médio de código usado para representar os dados da imagem original.

Os algoritmos dessas duas técnicas de codificação podem ser usados para codificação entrópica em qualquer dos processos de codificação [1].

Neste capítulo, discutimos a técnica de codificação de *Huffman* e a codificação aritmética que podem ser empregadas para comprimir a extensão de alfabetos, ou seja a codificação aritmética estudada neste capítulo é *não binária*. Alguns exemplos são usados para esclarecer a idéia do algoritmo de codificação.

3.2 Codificação de *Huffman*

A codificação de *Huffman* foi desenvolvida em 1952 e está sendo usada como código ótimo de tamanho variável para muitas aplicações.

Conseqüentemente, a filosofia dessa técnica é remover a redundância entre os dados sucessivos e codificar somente a informação nova.

Como já foi visto anteriormente, a eficiência da transmissão e o armazenamento de informação são fatores importantes. A codificação de *Huffman* é um dos métodos que pode ser empregado para aliviar uma parte do problema de armazenamento de dados e de transferência de informações, através da representação de dados por códigos mais eficientes.

A codificação de *Huffman* se beneficia das probabilidades de ocorrência de *pixels*, de maneira que os códigos menores podem ser usados para representar os *pixels* de ocorrência mais freqüente, já que os códigos maiores são usados para representar os *pixels* encontrados com menos freqüência. Isto é, a codificação de *Huffman* é uma das diversas técnicas onde os *pixels* são codificados com base nas probabilidades de ocorrência.

Os códigos de *Huffman* possuem a propriedade de prefixo, ou seja, impõe-se que nenhum código seja duplicado como o início de um código maior [13], onde os *pixels* são ordenados com respeito as suas probabilidades. Isso implica que o código de *Huffman* seja decifrável de forma única e assim proporcionando a possibilidade de decodificação instantânea. Portanto, pode-se aplicar um algoritmo de decodificação (descompressão) apropriado para restaurar os *pixels* em sua forma original.

Para entender o privilégio dessa propriedade considere como exemplo a seguinte situação: *Suponha um alfabeto de cadeia de quatro símbolos {x, y, z, w}, codificado como a seguir:*

$$\mathbf{x = 0} \quad , \quad \mathbf{y = 10} \quad , \quad \mathbf{z = 110} \quad , \quad \mathbf{w = 111}$$

Se a mensagem recebida pelo algoritmo de decodificação for "0100111110100", então esta seqüência poderia ser representada como ilustrada na figura (3.1).

mensagem codificada	0	10	0	111	110	10	0
mensagem decodificada	x	y	x	w	z	y	x

Fig. 3.1 - Propriedade de decodificação instantânea

Suponha que o alfabeto de quatro símbolos {x, y, z, w} têm as seguintes freqüências relativas de ocorrência (probabilidades de ocorrência): 1/2 , 1/4 , 1/8 e 1/8 respectivamente. O código de *Huffman* pode ser desenvolvido através da utilização de uma estrutura de árvore, como ilustrado na figura (3.2).

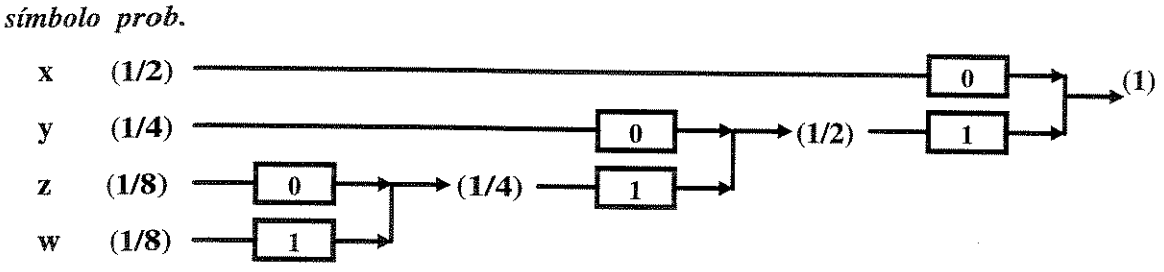


Fig. 3.2 - Construção dos códigos de Huffman para símbolos binários

O procedimento de codificação pode ser efetuado da seguinte maneira:

1. Primeiramente, os símbolos são listados em ordem decrescente de acordo com as probabilidades;
2. As duas menores probabilidades (probabilidades de z e w) são somadas para formar somente uma probabilidade;
3. Atribui-se ao valor "0" binário para o ramo superior e ao valor "1" binário para o ramo inferior (ou vice-versa, para cada combinação ou conjunto);
4. Reordena-se as probabilidades em ordem decrescente novamente;
5. Repetem-se os passos de 2 a 4 até obter uma probabilidade de ocorrência unitária;

6. O código de cada símbolo é obtido ao rastrear os ramos a partir da probabilidade unitária em direção ao símbolo escrevendo os valores "0's" e "1's" binários, os quais são encontrados no caminho de volta ao símbolo.

A tabela (3.1) ilustra a palavra código ("***codeword***") de cada símbolo, o comprimento da palavra código de cada símbolo (L_j) e a probabilidade-fração-binária (P) correspondente do comprimento da palavra código de cada símbolo. Note-se que a probabilidade P de símbolo j é igual a dois elevado a L_j negativo ($P = 2^{-L_j}$).

Tab. 3.1 - Códigos de Huffman

símbolo	*palavra código*	*comprimento* (L_j)	*probabilidade* (P)
x	0	1	0.100
y	10	2	0.010
z	110	3	0.001
w	111	3	0.001

Utilizando o procedimento anterior, um grupo de símbolos "y y z x" pode ser codificado como "10101100".

Entretanto, o processo de decodificação, como exemplo, da mesma cadeia de símbolos será feito comparando-se os *bits* da palavra código com os das palavras código de símbolos. Portanto, o primeiro símbolo que foi codificado é "y", pois não tem palavra código que começa com *bit* "1" e a única palavra código que começa com *bits* "10" é o símbolo "y". Tiram-se os *bits* "10" da palavra código original, e o resto se compara com as palavras código de símbolos novamente obtendo-se o segundo símbolo que será "z". Finalmente, a palavra código "0", aquela que sobrou, pertence ao símbolo "x".

De acordo com o que foi mencionado anteriormente, os algoritmos de codificação e decodificação de *Huffman* são realizados procurando-se os códigos de qualquer conjunto de probabilidade dado numa tabela montada correspondente de comprimento L_j do palavra código de cada símbolo.

3.3 Codificação Aritmética

A codificação aritmética é considerada como uma técnica poderosa comparada com outras técnicas de compressão. Além disso, ela determina as palavras código durante o processo de codificação.

Sendo assim, o codificador aritmético substitui os *pixels* de ocorrência freqüente por um código binário curto, enquanto os *pixels* encontrados com menos ocorrência são substituídos por um código binário mais longo.

O primeiro trabalho em codificação aritmética foi feito por **SHANNON** em 1948 [9]. Sua idéia foi a distribuição das mensagens em ordem de suas probabilidades. Em seguida, enviam-se as probabilidades cumulativas (P_C) de cada símbolo codificado (corrente de código) em fração binária e decodifica-se pela operação de comparação. Em seguida, **SHANNON** e **ELIAS** encontraram o problema de precisão, ou seja, toda vez que a mensagem aumenta no comprimento, é requerido um crescimento na precisão. Para aliviar esse problema eles usaram comprimento fixo para os códigos. Em 1972, surgiu uma técnica desenvolvida por **SCHALKWIJK** [16], o código foi de tipo ***LIFO*** (***Last In First Out***), ou seja, o último *pixel* codificado é o primeiro decodificado. **RISSANEN** [17] descreveu a eficiência da técnica de codificação aritmética em que se evita o código de bloco; o código foi de tipo *LIFO*. **RISSANEN** conseguiu aliviar o problema de precisão pelas aproximações no projeto de código aritmético de *LIFO* (usando a precisão fixa). **PASCO** [18] descobriu a forma de código aritmético do tipo ***FIFO*** (***First In First Out***), isto é, o primeiro *pixel* codificado é o primeiro decodificado; o algoritmo armazena a corrente de códigos na memória de computador até o último *pixel* codificado e além disso, utilizou-se a idéia do **RISSANEN** para controlar o problema de precisão. **RISSANEN** e **LANGDON** [19] apresentaram um trabalho que aproveitou o código aritmético de *FIFO*; o objetivo do trabalho foi generalizar o conceito de codificação aritmética para adotar outros tipos de códigos: códigos sem blocos ("*nonblock codes*"), como código de **ELIAS** [20], [21]. Em anos seguintes, a derivada dos conceitos básicos de técnica da codificação aritmética binária foi apresentada por **RISSANEN**, **PASCO** e **LANGDON**. Essa técnica utiliza a estimação de

probabilidade que deriva das renormalizações de intervalo. No caso do padrão *JBIG* que é estudado no capítulo 5, a estimativa de probabilidade é feita consultando-se uma tabela construída pelo próprio *JBIG*.

Devido à taxa de transmissão e o fator de compressão da codificação aritmética serem melhores que aqueles utilizados na codificação de *Huffman*, essa codificação tem merecido a atenção dos pesquisadores de processamento digital de sinais (imagens).

3.3.1 Algumas Considerações na Codificação Aritmética

Duas considerações principais para a codificação aritmética são:

- A palavra código do *pixel* é a probabilidade cumulativa P_C ; cada palavra código é a soma das probabilidades relativas P dos *pixels* antecedentes. Pegando o mesmo exemplo da seção (3.2) de quatro símbolos {x,y,z,w}, e reordenando-se como está ilustrado na tabela (3.2), pode-se calcular a P_C de cada símbolo;

Tab. 3.2 - Codificação Aritmética

símbolo	*comprimento* (L_j)	*probabilidade relativa (P)*	*probabilidade cumulativa* (P_C)
w	3	0.001	0.000
y	2	0.010	0.001
x	1	0.100	0.011
z	3	0.001	0.111

- O algoritmo de codificação aritmética divide um intervalo da linha dos números reais entre "0" e "1" correspondente às palavras código, ou seja, as probabilidades cumulativas de símbolos são distribuídas dentro de um intervalo [0,1), chamado **intervalo de codificação corrente** (A_C). As probabilidades relativas ao valor de cada símbolo são distribuídas dentro de um intervalo [0,1), onde o colchete "[" e

o parênteses ")" indicam se o dígito pertence ou não pertence ao intervalo, respectivamente. A figura (3.3) mostra a subdivisão do intervalo.

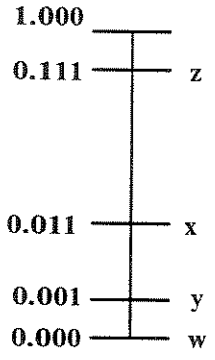


Fig. 3.3 - Palavras código dentro a unidade de intervalo

Como está ilustrado na figura (3.3), o tamanho do sub-intervalo que está acima de cada palavra código é equivalente à probabilidade relativa P de ocorrência do próprio símbolo. Em outras palavras, o intervalo de cada símbolo começa a partir da sua palavra código e termina com o início da palavra código do símbolo seguinte. Por exemplo, o intervalo do símbolo "y" é [0.001 , 0.011).

3.3.2 O Processo de Codificação

De acordo com as duas considerações que foram frisadas anteriormente, a codificação aritmética codifica/decodifica um símbolo por iteração.

O algoritmo de codificação aritmética divide o intervalo A_C em sub-intervalos com tamanhos proporcionais à probabilidade relativa P de ocorrência do símbolo. O intervalo que está associado ao valor do símbolo a ser codificado é denominado **novo intervalo de codificação corrente** (A_N). Essa operação é feita de maneira recursiva, ou seja, o intervalo A_N é novamente dividido em sub-intervalos na mesma forma que o intervalo A_C foi dividido e renomeado para intervalo A_C , e assim por diante até o algoritmo codificar todos os símbolos.

Durante o processo de codificação, é necessário especificar o intervalo A_C . O codificador aritmético mantém o valor do tamanho do intervalo numa variável chamada A_C e sua base (palavra código) numa variável chamada C_C . Essa base seria o limite inferior do intervalo.

Através do tamanho e da base do intervalo A_C , o codificador aritmético determina o novo intervalo A_N e a nova base C_N .

A figura (3.4) mostra a subdivisão de um intervalo com um exemplo de uma seqüência de símbolos a ser codificada "x x y".

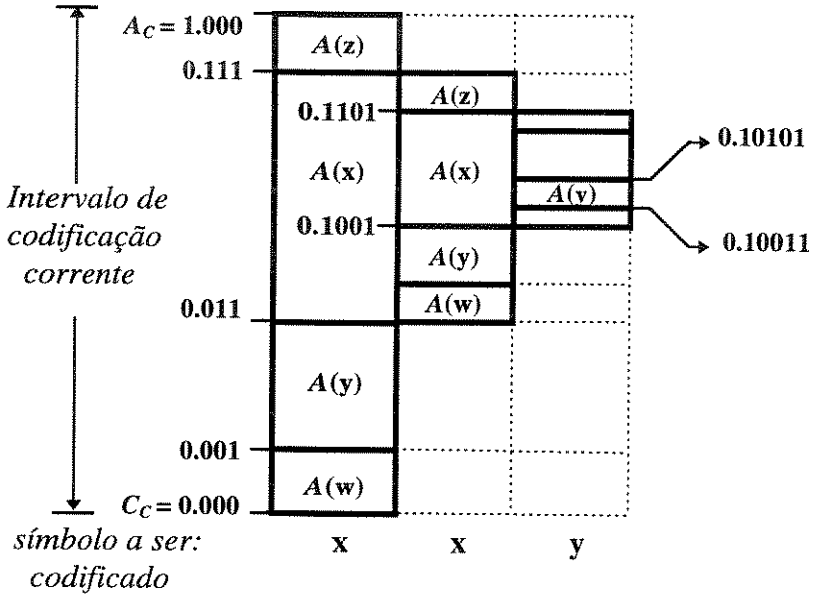


Fig. 3.4 - Subdivisão do intervalo para codificar a seqüência "x x y"

Para cada símbolo a ser codificado deve-se determinar a sua palavra código e o tamanho do seu intervalo. Portanto, para determinar a palavra código nova , nós usamos a seguinte equação:

$$C_N = C_C + (A_C * P_C) \qquad (3.1)$$

onde:

C_N : Palavra código nova;

C_C : Palavra código corrente;

A_C : Tamanho do intervalo de codificação corrente;

P_C : Probabilidade cumulativa do símbolo sendo codificado.

Para determinar o intervalo de codificação novo (tamanho) nós usamos:

$$A_N = A_C * P \tag{3.2}$$

onde:

A_N : Tamanho novo do intervalo de codificação;

P : Probabilidade relativa do símbolo sendo codificado.

Para explicar o procedimento de codificação aritmética, considerando a mesma seqüência de símbolos a ser codificada "x x y" usa-se a tabela (3.2) para consultar a probabilidade de símbolo. O código aritmético usado é do tipo *FIFO*.

O processo de codificação aritmética é resumido no seguinte:

- Primeiro, o intervalo A_C é dividido em sub-intervalos correspondentes da probabilidade cumulativa P_C de cada símbolo. O valor inicial do intervalo A_C é "1.000" e da palavra código corrente C_C é "0.000". O primeiro símbolo a ser codificado é "x", sendo que para se determinar os valores novos as eqs. (3.1) e (3.2) são usadas as relações:

$$C_N = C_C + (A_C * P_C)$$
$$= 0.000 + (1.000 * 0.011) = 0.011$$

$$A_N = A_C * P$$
$$= 1.000 * 0.100 = 0.100$$

O símbolo "x" tem palavra código "0.011" e intervalo [0.011 , 0.111). Note-se que o intervalo começa com o valor C_N ;

- Para codificar o segundo símbolo "x", o novo intervalo A_N será o intervalo A_C e a palavra código nova C_N será a palavra código C_C . O intervalo A_C se divide usando-se o mesmo número de subdivisões que foi usado para a divisão do intervalo original, ou seja, A_C é dividido em sub-intervalos relacionados às probabilidades P_C [7]. Os valores novos do intervalo A_N e da palavra código C_N são determinados em seguida conforme mostrado:

$$C_N = 0.011 + (0.100 * 0.011) = 0.1001$$

$$A_N = 0.100 * 0.100 = 0.010$$

Agora, o símbolo "x" tem a palavra código "0.1001" e intervalo [0.1001 , 0.1101);

- Repetindo-se o mesmo processo de conversão de valores novos pelos valores correntes, a operação recursiva para codificar o terceiro símbolo "y" é:

$$C_N = 0.1001 + (0.010 * 0.001) = 0.10011$$

$$A_N = 0.010 * 0.010 = 0.0001$$

O símbolo "y" tem a palavra código "0.10011" e intervalo [0.10011 , 0.10101).

Então, segundo a codificação aritmética da seqüência de símbolos "x x y", o intervalo obtido é [0.10011 , 0.10101). Qualquer valor dentro esse intervalo (maior ou igual ao valor "0.10011" e menor do que o valor "0.10101") servirá para identificar o intervalo e representar o código a ser transmitido ou armazenado como palavra código

da seqüência de símbolos original. A subdivisão do intervalo desse exemplo é mostrada na figura (3.4).

A palavra código é tratada como uma magnitude. Ela é uma fração binária que aponta para um número real dentro do intervalo de codificação corrente.

3.3.3 O Processo de Decodificação

O decodificador aritmético usa o mesmo modelo usado no codificador aritmético. O processo de decodificação pode ser feito por comparação de magnitude sendo que o sub-intervalo é apontado pela palavra código [2].

O procedimento de decodificação aritmética usa a palavra código e o tamanho do intervalo do mesmo modo que o procedimento de codificação aritmética. Primeiro, o limite do intervalo é identificado e em seguida pode-se determinar o símbolo codificado.

Basicamente, a função do decodificador aritmético é subtrair qualquer intervalo que foi somado com a cadeia do código pelo codificador aritmético. Essa operação é efetuada recursivamente para cada símbolo a ser decodificado, usando-se o processo de subdivisão de intervalo que foi usado para a codificação aritmética. Isso significa que o decodificador aritmético desfaz o que foi feito pelo codificador aritmético recursivamente.

Como exemplo, vamos explicar o procedimento de decodificação aritmética através da palavra código obtida pela operação de codificação aritmética. Vamos então descobrir os símbolos originais. A palavra código foi "0.1010011" e o intervalo foi [0.10011 , 0.10101). Usando-se a tabela (3.2) para comparar o intervalo com o valor de probabilidade cumulativa P_C de cada símbolo, tem-se:

- o codificador aritmético compara a palavra código "0.1010011" com os valores de probabilidade cumulativa P_C . Para codificar o símbolo "y", a cadeia de código teria um valor dentro do intervalo [0.001 , 0.011), evidentemente, o símbolo "y" não é o

primeiro símbolo que foi codificado. Para codificar o símbolo "x", a cadeia de código teria um valor pertencente ao intervalo [0.011 , 0.111), portanto, esse intervalo contem a palavra código "0.1010011", ou seja, a palavra código ficaria dentro do intervalo do símbolo "x". Isso indica que, o símbolo "x" é que deve ser o primeiro símbolo da seqüência de símbolos que foi codificada;

- o valor de probabilidade cumulativa P_C do símbolo "x" é subtraído da palavra código pelo decodificador aritmético, e tem-se:

0.1010011 - 0.011 = 0.0100011

- no codificador aritmético, o segundo sub-intervalo foi ajustado multiplicando-se pelo valor 0.1 . Por outro lado o decodificador aritmético desfaz esse passo. O resultado da operação de subtração é dividido pelo mesmo valor 0.1, e tem-se:

0.0100011 / 0.1 = 0.100011

De maneira recursiva, o decodificador aritmético repete a mesma operação anterior para se obter o segundo símbolo que foi codificado. Tem-se:

- a palavra código obtida é "0.100011". Com a operação de comparação, a palavra código ficaria dentro do intervalo [0.011 , 0.111), o qual pertence ao símbolo "x". Portanto, o símbolo "x" deve ser o segundo símbolo da seqüência de símbolos que foi codificada;

- subtraindo-se o valor "0.011" da palavra código, temos:

0.100011 - 0.011 = 0.001011

- dividindo-se o resultado pelo valor 0.1, tem-se:

0.001011 / 0.1 = 0.01011

Para obter o terceiro símbolo que foi codificado pelo codificador aritmético, basta reaplicar o primeiro passo. O decodificador aritmético compara a palavra código obtida com os valores de probabilidade cumulativa P_C. Portanto, o terceiro símbolo que foi codificado é "y".

Capítulo 4

MODELO ESTATÍSTICO

4.1 Considerações Especiais no Sistema de Compressão

O processo de compressão consiste na transformação dos dados para uma representação codificada, a qual é transmitida ou armazenada.

Para qualquer técnica de compressão, principalmente a codificação aritmética, existem três unidades principais denominadas: o modelo, a estimação de estatística (probabilidade) do *pixel* e o codificador.

4.1.1 O Modelo

O modelo proporciona as características sobre os dados e sobre o *pixel* a ser codificado. Determina o *pixel* atual a ser codificado e seu contexto (*CX*). Essa operação (geração do contexto) é feita sucessivamente para cada *pixel*. O modelo pode ser separado da unidade do codificador.

4.1.2 O Contexto

O contexto é um número inteiro calculado pela seqüência de *pixels* antecedentes ao *pixel* a ser codificado. É um estado determinado pelos *pixels* passados sendo utilizado tanto para o codificador como para o decodificador.

4.1.3 Estimação de Estatística

O método de estimação calcula a distribuição da freqüência relativa (probabilidade) usada para cada contexto. Essa probabilidade é usada pela unidade do codificador. O cálculo das probabilidades pode ser feito antes do processo de codificação, como é feito na técnica de codificação de *Huffman*, ou durante o processo de codificação como na técnica de codificação aritmética.

4.1.4 Unidade do Codificador

O codificador recebe o *pixel* a ser codificado e indicações sobre sua probabilidade relativa, por exemplo o contexto, e gera a cadeia de códigos (**palavras código**).

4.2 Tipos de Modelos

Vários tipos de modelos são usados nos sistemas de compressão. Esses modelos dependem da aplicação usada. No modelo simples chamado **sem memória** ("*memoryless*"), os *pixels* são codificados de acordo com um código único (como na codificação de *Huffman*).

O modelo para os dados fonte às vezes especializa-se na formulação da função recursiva para se obter uma fórmula da "***Finite State Machine*** (***FSM***)". O modelo *FSM* é a generalização de diversas versões de modelos ***Markov*** para processos estocásticos, onde a probabilidade do *pixel* a ser codificado é dependente de um número fixo (contexto) dos valores observados previamente, como por exemplo, o modelo de *Markov* de primeira ordem, que usa o *pixel* prévio do *pixel* a ser codificado como o contexto.

O modelo usado neste trabalho é o modelo estacionário adaptativo. Esse modelo é *FSM* para documentos de imagens em 2 *tons* (*B/W*). A adaptação nesse modelo é realizada com a ocorrência dos *pixels* (as probabilidades associadas com os contextos) enquanto a seqüência de *pixels* está sendo codificada, isto é, a probabilidade de cada *pixel* é determinada durante o processo de codificação. Essa probabilidade muda de um estado para o próximo que tenha mais informação sobre a seqüência de *pixels* [2].

4.3 "*Template*" Modelo

Como foi referido anteriormente, o modelo usado é o *FSM* adaptativo. Esse é o modelo de *Markov* de décima ordem, em outras palavras, o "*template*" modelo contém 10 *pixels* vizinhos e circundantes ao *pixel* a ser codificado. Os *pixels* circundantes são da mesma linha e das linhas prévias do *pixel* atualmente codificado. Esses *pixels* têm dependência forte com o *pixel* que está sendo codificado e formam o contexto. Nesse caso *o contexto será um valor inteiro entre* 2^{10} *valores diferentes*, onde o expoente 10 é um número inteiro dos *pixels* prévios do *pixel* a ser codificado.

4.4 Estrutura de Modelo

A estrutura de modelo ("***template*** **modelo**) dos contextos e dos *pixels* a serem codificados é construída em várias formas. O padrão *JBIG* fornece duas formas de "*templates*":

- Uma para codificar a menor camada de resolução (modo progressivo) e serve também para codificar a imagem sem usar o modo progressivo, ou seja, usa o modo seqüencial. Esse "*template*" pode ser composto de duas ou de três linhas;

- Outra para codificar as camadas de resolução diferenciais (modo progressivo) [2].

Neste trabalho usamos a estrutura de modelo de duas linhas para a codificação/decodificação sem usar modo progressivo. As figuras (4.1a) e (4.1b) mostram a estrutura de modelo de duas e três linhas, respectivamente.

	X(t-M-3)	X(t-M-2)	X(t-M-1)	X(t-M)	X(t-M+1)	X(t-M+2)
X(t-3)	X(t-2)	X(t-1)	X(t)	X(t+1)		

Fig. 4.1a - Estrutura de modelo de duas linhas

	X(.)	X(.)	X(.)	
X(.)	X(.)	X(.)	X(.)	X(.)
X(.)	X(.)	X(t+1)		

Fig. 4.1b - Estrutura de modelo de três linhas

onde:

x(.) : O *pixel* que faz parte do "*template*" modelo além do *pixel* x(t+1);

t : Posição do *pixel* x(t), codificado em modo seqüencial ("*raster mode*");

t+1 : Posição do *pixel* x(t+1) a ser codificado; não faz parte do "*template*" modelo;

M : Número de *pixels* por linha (documentos do *CCITT*, M = 1728);

x(t+1): O *pixel* acima do *pixel* sendo codificado.

O "*template*" modelo é separado da codificação aritmética adaptativa. Cada *pixel* no "*template*" modelo é correspondente a um *bit* no contexto. Esse contexto será usado pelo codificador aritmético adaptativo *CAA* para codificar o *pixel* a ser codificado, também usado pelo decodificador aritmético adaptativo *DAA* para recuperar o *pixel* original.

Para calcular o contexto, os *pixels* do "*template*" modelo são compartilhados para se determinar o valor do contexto associado ao *pixel* que está sendo codificado.

Temos:

$$CX = \mathbf{x(t\text{-}M\text{-}3)\ x(t\text{-}M\text{-}2)\ .....\ x(t\text{-}M\text{+}2)\ x(t\text{-}3)\ .....\ x(t)} \qquad \mathbf{(4.1)}$$

A eq. (4.1) proporciona um valor inteiro (*CX*) entre 1024 valores possíveis para o contexto (*CX*).

Capítulo 5

CODIFICAÇÃO ARITMÉTICA BINÁRIA ADAPTATIVA (*ABAC* - *Adaptive Binary Arithmetic Coding*)

5.1 Introdução

O advento da codificação aritmética proporcionou novas aplicações para o processamento de imagem. Portanto, tem merecido a atenção dos especialistas nessa área. O método de codificação aritmética utiliza a estatística das mensagens (*pixels*), ou seja, as probabilidades de ocorrência dos *pixels* de fontes discretas. A codificação aritmética pode ser tratada como uma técnica de codificação binária adaptativa (compressão de dados binários), denominada **codificação aritmética binária Adaptativa** (***ABAC***). Essa codificação é também uma técnica de compressão poderosa e exata, que tem atraído a atenção dos pesquisadores nos últimos anos. Além disso, proporciona maior flexibilidade e melhor eficiência comparada à codificação de *Huffman*. O procedimento de codificação é recursivo, isto é, opera codificando e decodificando uma amostra de dados em cada iteração. A *ABAC* é uma das famílias de códigos que compartilha a propriedade de tratar um fluxo de *pixel* codificado como uma magnitude. Os dados originais podem ser resgatados do fluxo de *pixel* codificado "*code string*" com a técnica de *FIFO*.

Neste capítulo a **codificação aritmética binária adaptativa** (*ABAC*) será estudada para aplicações em imagens de 2 *tons* baseando-se em técnicas de compressão padrão *JBIG*.

5.2 Princípio de Codificação Aritmética Binária Adaptativa (*ABAC*)

A codificação aritmética binária adaptativa *ABAC* que é utilizada neste trabalho é a codificação entrópica. Nessa codificação trata-se os *pixels* da imagem em 2 *tons* como se tivessem dois níveis, isto é, cada *pixel* tem um valor binário "1" (**preto**) ou "0" (**branco**). O procedimento de *ABAC* baseia-se numa *tabela de estimação de probabilidade* [apêndice A] para estimar a probabilidade condicional relativa do valor do *pixel* corrente que será codificado. Esse tipo de codificação tem como características a simplicidade e a conscientização segundo o padrão *JBIG*.

Na *ABAC* que utiliza o modo progressivo tem-se em cada tira de cada camada de resolução, o sub-bloco do codificador aritmético adaptativo (***CAA***) recebe 4 fluxos de dados de entrada com as seguintes definições: *pixel* (***PIX***), contexto (***CX***), valor típico de predição (***VTP***) e valor determinístico de predição (***VDP***). O *CAA* produz na saída somente um fluxo de dados definido como *pixels* codificados de tira ou palavras código de tira (***PCT***). A codificação é feita seqüencialmente pegando-se um *pixel* de cada fluxo de entrada.

Entretanto, os *pixels* de saída são gerados de acordo com o procedimento utilizado pelo *CAA*. A figura (5.1) mostra a entrada e a saída do sub-bloco de *CAA* com modo progressivo.

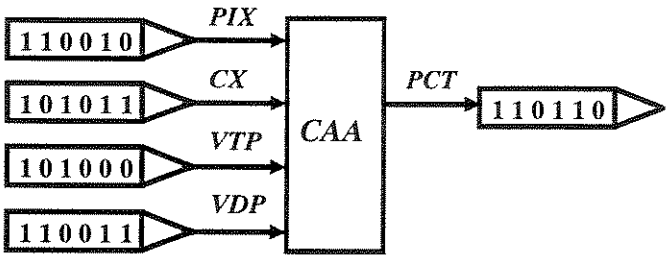


Fig. 5.1 - Entrada e saída do sub-bloco de CAA com modo progressivo

Neste trabalho a codificação utiliza somente dois fluxos de entrada: o *pixel* (*PIX*) e o contexto (*CX*), como é mostrado na figura (5.2). O tratamento será com uma única camada de resolução sem referência às outras camadas, ou seja, trata-se a imagem total como se fosse formada de uma única camada de resolução e sem dividí-la em tiras.

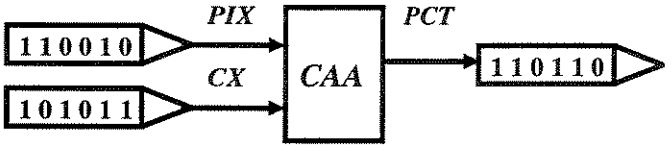


Fig. 5.2 - Entrada e saída do sub-bloco de CAA sem modo progressivo

5.2.1 Conceitos Fundamentais de *ABAC*

Semelhante às seções (3.3.1) e (3.3.2), no processo de *ABAC* o fluxo de entrada *PIX* é mapeado para um número real **x** dentro um intervalo [0,1), chamado **intervalo de codificação corrente** (*A*). O fluxo de saída (*PCT*) que vai ser transmitido ou armazenado é a expansão binária do número **x**.

O intervalo *A* é dividido em dois sub-intervalos com tamanhos proporcionais à probabilidade relativa de ocorrência do *PIX*. O intervalo que está associado ao valor do *PIX* a ser codificado é denominado **novo intervalo de codificação corrente** (*A*(.)). O intervalo *A*(.) é novamente dividido em dois sub-intervalos e renomeado para intervalo *A*, assim sucessivamente.

Durante o processo de codificação, o *CAA* utiliza o valor do tamanho do intervalo de codificação corrente, que está numa variável chamada *A* e o valor da base do intervalo *A*, que está numa variável *C*. O *CAA* determina o novo intervalo *A*(.) e sua base *C*(.) usando as duas variáveis *A* e *C*. A figura (5.3) mostra a subdivisão de um intervalo, usando-se como exemplo, uma seqüência de *pixels* (fluxo *PIX*) a ser codificada dada por {1, 0, 0, 1}.

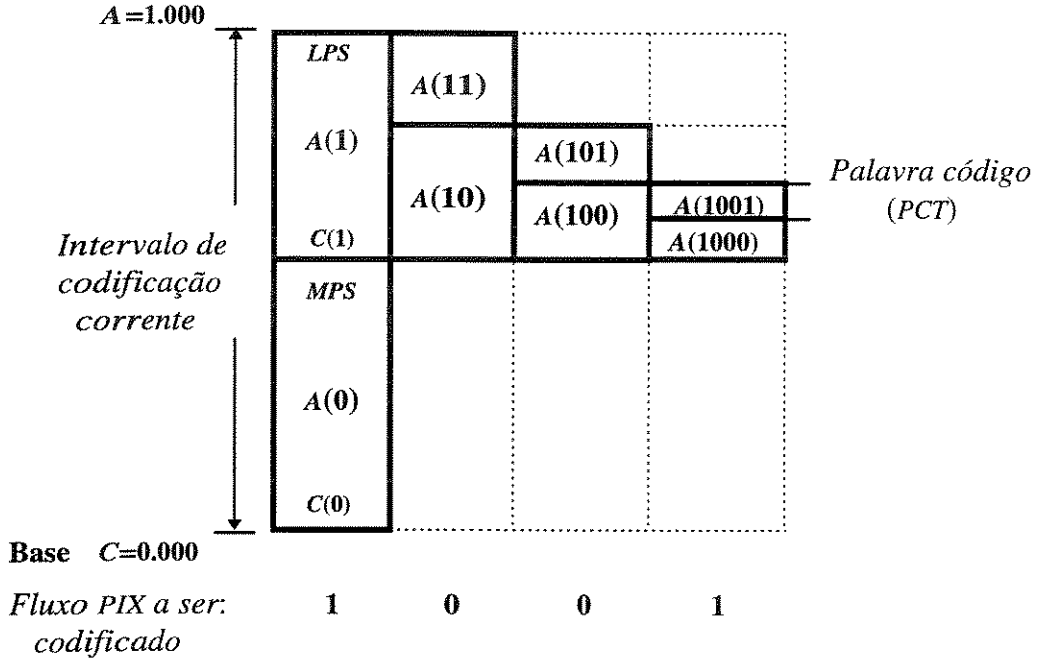


Fig. 5.3 - Operação de subdivisão do intervalo

O fluxo de saída (*PCT*) que vai transmitir ou armazenar será obtido pela variável da base *C*(.). O valor de *C*(.) é assume qualquer valor dentro o intervalo *A*(.).

5.2.2 Ferramentas de Operação

As operações de codificação podem ser realizadas usando-se precisão fixa para um número aritmético inteiro. Para tanto são necessários dois registradores: o **registrador** *A* que contém o tamanho do intervalo *A* normalizado e o **registrador** *C* que contém a base do intervalo *A*. O tamanho é mantido na faixa de [0x8000, 0x10000], onde "0x" indica que o número inteiro é hexadecimal. Quando o tamanho é inferior ao limite 0x8000 , o valor se duplica recursivamente até ficar maior ou igual ao limite 0x8000. Essas duplicações são chamadas renormalizações. Toda vez que o registrador *A* é duplicado, o registrador *C* também é duplicado.

- A tabela (5.1) mostra a estrutura de registradores de 32 *bits* usada na implementação do *CAA*;

- A tabela (5.2) mostra a estrutura de registradores usada na implementação do *DAA*. Os registradores C_H e C_L trabalham como se fossem um registrador de 32 *bits*.

Tab. 5.1 - Estrutura de registradores do codificador

bits / registrador	**MSB**			**LSB**
A	00000000	0000000T	TTTTTTTT	TTTTTTTT
C	0000cppp	ppppppss	bbbbbbbb	bbbbbbbb

Tab. 5.2 - Estrutura de registradores do decodificador

bits / registrador	**MSB**			**LSB**
A	00000000	0000000T	TTTTTTTT	TTTTTTTT
C_H	----------	----------	bbbbbbbb	bbbbbbbb
C_L	----------	----------	pppppppp	00000000

onde:

T : *Bits* fracionais do valor de intervalo *A*;

b : *Bits* fracionais do valor de base de intervalo *A*;

s : *Bits* necessários para *carry-over* (espaço de *bits*);

p : *Bits* de posições de dados que têm sido transformados do registrador *C*;

c : *Bit* de *carry*.

- Para impedir que o *overflowing* do registrador *C* aconteça durante a operação, um *buffer* externo (***BUF***) é necessário para receber um *byte* de dados referentes aos *bits* mais significativos (**MSB**) do registrador *C*. Esse *BUF* pode ocupar uma área da memória ou um arquivo no disco, onde são armazenados os códigos de fluxo de saída (*PCT*). O resultado final (**palavras código**) do procedimento de *CAA* correspondente à informação de uma imagem total, transmitido ou armazenado é o conteúdo do *BUF*;

- **Contador** *CT* de deslocamentos de *bits*: a função básica é indicar quando um *byte* do *BUF* está pronto para ser transmitido ou armazenado durante o processo de codificação, ou quando deve ser lido um *byte* do fluxo *PCT* durante o processo de decodificação;

- Espaço de memória usado como uma pilha apontada pelo **contador** *SC*: nela a rotina armazena números de *bytes* de valor 0xff ou 0x00 até resolver o problema de *carry-over*;

- Uma tabela foi elaborada pelo padrão *JBIG* para os estados de estimativa de probabilidade dos valores do fluxo *CX* [apêndice A];

- A variável ***ST[CX]*** de ***7bits*** para os estados de estimativa de probabilidade dos valores do *CX* e ***1bit*** para o valor de *PIX* de maior ocorrência denominado ***VMPS[CX]***.

Juntas essas variáveis capturam a estimativa de probabilidade adaptativa associada com o valor do *CX* particular.

5.3 Processo de *ABAC*

Inicialmente, o procedimento de *CAA* considera que *pixels* com valor branco "0" são de maior ocorrência do que *pixels* com valor preto "1".

Os dois sub-intervalos que o procedimento de *CAA* gera durante o processo de codificação são:

a) Sub-intervalo de *pixel* de maior ocorrência ***MPS*** (***"more probable symbol"***);

b) Sub-intervalo de *pixel* de menor ocorrência ***LPS*** (***"less probable symbol"***).

O sub-intervalo de *LPS* é colocado acima do sub-intervalo de *MPS* e sempre com tamanho menor. Portanto, durante o processo de codificação de *ABAC*, o procedimento de *CAA* mapeia cada *pixel* oferecido pelo fluxo *PIX* para um número real dentro do intervalo *A*. Esse número real é equivalente à estimativa de probabilidade adaptativa do *pixel* oferecido.

O intervalo *A*(.) que está sendo codificado pelo procedimento de *CAA* é renomeado para o intervalo *A* . Em seguida, o processo de *CAA* recomeça novamente.

5.3.1 Determinação dos Intervalos

Uma aproximação aritmética simples é usada na subdivisão de intervalo. Para calcular um sub-intervalo de *LPS* é necessário realizar uma multiplicação de um intervalo *A* com uma estimativa de probabilidade (p_e) do sub-intervalo de *LPS* como ilustrado na eq. (5.1).

$$p_e * A = LSZ \tag{5.1}$$

Se a quantidade armazenada em *LSZ* for interpretada como sendo a probabilidade p_e, então, *LSZ* é igual ao tamanho do sub-intervalo de *LPS*. Assim, tem-se:

$$p_e \cong LSZ \tag{5.2}$$

Essa interpretação via eq. (5.1) é válida porque o valor decimal do intervalo *A* fica dentro da faixa de [0x8000, 0x10000] , a qual se aproxima do valor unidade. Porisso é necessário renormalizar o tamanho do intervalo *A* caso o limite seja inferior a 0x8000.

O valor de ***LSZ*** *é um valor da tabela de estimação de probabilidade[apêndice A] relativo ao estado de estimativa de probabilidade do valor do CX particular* (***LSZ[ST[CX]]***).

Quando o procedimento de *CAA* codifica o sub-intervalo de *LPS*, uma soma é realizada (valor do sub-intervalo de *MPS* mais a base (registrador *C*)) como está ilustrado na eq. (5.3). O intervalo *A* (registrador *A*) é reduzido para o intervalo *A*(.), associado ao valor do sub-intervalo *LPS* (*LSZ*), como descrito na eq. (5.4).

Tomando o primeiro intervalo da figura (5.3) como exemplo, aplicam-se as seguintes equações:

$$C(1) = A(0) + C(0) \tag{5.3}$$

$$A(1) = LSZ \tag{5.4}$$

e

$$A(0) = A - A(1) \tag{5.5}$$

$$C(0) = C \tag{5.6}$$

onde:

A(0) : Tamanho do sub-intervalo de *MPS*;

A(1) : Tamanho do sub-intervalo de *LPS*;

$C(0)$: Base do sub-intervalo de *MPS*;

$C(1)$: Base do sub-intervalo de *LPS*.

Enquanto o sub-intervalo de *MPS* está sendo codificado, o valor do intervalo A é reduzido para o intervalo $A(.)$, associado ao valor do sub-intervalo de *MPS*, o qual é igual ao tamanho do intervalo A menos o valor do sub-intervalo de *LPS*, como está ilustrado na eq. (5.5). O valor da base ficará sem mudança, como descrito na eq. (5.6).

Depois que essas operações são realizadas, se o valor do registrador A é inferior ao limite 0x8000 é necessário renormalizá-lo. Isso é feito para que o valor do registrador A fique dentro da faixa permitida. Enquanto isso o registrador C também é renormalizado.

Durante o processo de codificação, devido à aproximação feita no processo de subdivisão do intervalo de probabilidade, pode acontecer que o tamanho do sub-intervalo de *LPS* seja maior que o tamanho do sub-intervalo de *MPS*. Para evitar essa *inversão* do tamanho do sub-intervalo, os dois sub-intervalos são trocados. Essa operação ocorre quando uma renormalização é necessária. A troca de tamanho dos sub-intervalos é denominada troca condicional (***MPS/LPS***).

Toda vez que uma renormalização ocorre, solicita-se um processo de estimação de probabilidade. Esse processo determina o novo estado de estimativa de probabilidade adaptativa do valor do *CX* atualmente codificado para se codificar o próximo *pixel*.

5.3.2 Rotinas e Fluxogramas do *CAA*

O procedimento do *CAA* é executado para cada *PIX* de imagem e para cada valor correspondente de *CX*. O procedimento de *CAA* utiliza 3 rotinas: a primeira rotina é a **INITC** para inicialização das ferramentas de operação e é chamada quando se inicia o processo de codificação. A segunda rotina é a **CODIF** para codificação da imagem *pixel*-a-*pixel* , ou seja, *bit*-a-*bit*. A terceira rotina é a **FIMC** é chamada na saída do

processo para encerrar a operação de *CAA*. A figura (5.4) mostra o fluxograma do *CAA*. Em todas as vezes o procedimento de *CAA* faz questão de testar se já se atingiu o fim das informações da imagem. Atingindo o fim, o procedimento passa para a rotina **FIMC**. Caso contrário, segue para ler um novo *PIX* e um novo *CX*, em seguida, chama a rotina **CODIF**.

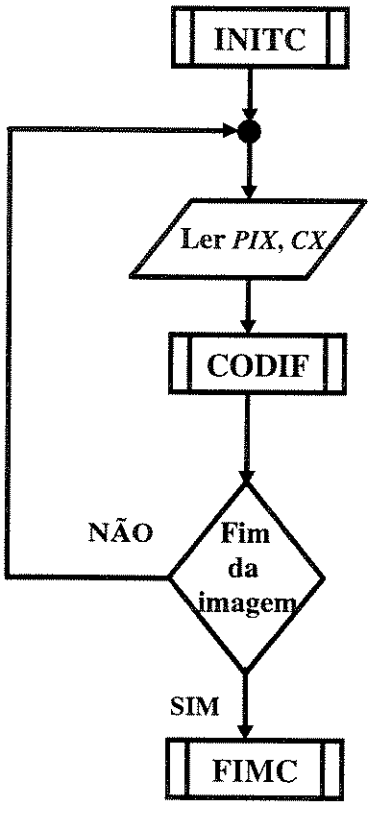


Fig. 5.4 - Fluxograma do CAA

A seguir passamos a descrever detalhadamente a função de cada rotina:

5.3.2a) Rotina INITC

A função básica dessa rotina é colocar os valores iniciais em cada registrador e em cada posição de memória.

Uma exigência da *CCITT* é que toda página ou documento precisa começar com um *pixel* de valor branco "0". Então, para todos os valores possíveis do *CX* deve-se

inicializar uma variável *VMPS[CX]* de 1 *bit* (*bit* **MSB** da variável *ST[CX]*) com o valor "0" para começar o processo de codificação sendo que os estados de estimação de probabilidade são inicializados com o valor "0" (primeiros 7 *bits* da variável *ST[CX]*).

O registrador *A* é inicializado com o valor "0x10000"; o registrador *C* com o valor "0"; o contador *CT* com o valor "11" (1 *byte* de p e 3 *bits* de s, conforme tabela (5.1)) e o contador de pilha *SC* com o valor "0". A figura (5.5) mostra o fluxograma da rotina **INITC**.

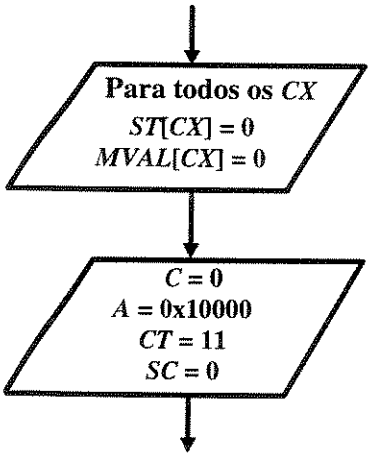


Fig. 5.5 - Fluxograma da rotina INITC

5.3.2b) Rotina CODIF

Essa rotina tem a função de verificar se o *PIX* corrente é igual ou não ao valor de *PIX* de maior ocorrência *MVAL[CX]*. Nesse caso, a verificação é feita todas as vezes que o *CAA* ler um *PIX* de fluxo de entrada, *bit*-a-*bit*. Caso sejam iguais, internamente essa rotina solicita a chamada da rotina **MPSCOD** para codificar o *PIX* de maior ocorrência. Caso contrário, a rotina prepara-se para **LPSCOD** a fim de gerar o processo de *CAA* onde se codifica o *PIX* de menor ocorrência. A figura (5.6) mostra o fluxograma da rotina **CODIF**.

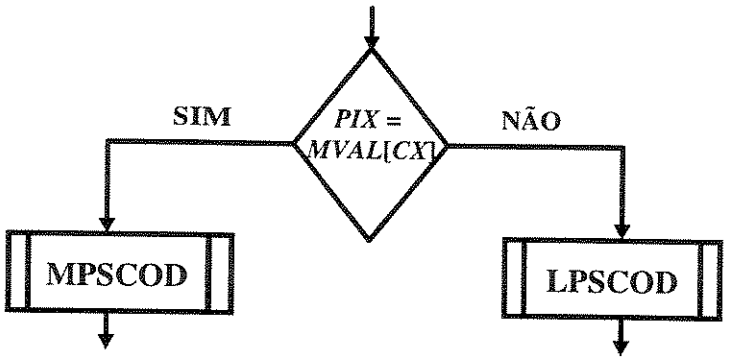


Fig. 5.6 - Fluxograma da rotina CODIF

5.3.2c) Rotina MPSCOD

Basicamente, a função dessa rotina é codificar qualquer *PIX* com maior ocorrência dentro do fluxo de dados binários de entrada.

O processo começa realizando-se uma operação de redução de tamanho do intervalo *A* como ilustrado anteriormente na eq. (5.5), produzindo-se o intervalo *A(MPS)* (*sub-intervalo de MPS*). Se o valor do intervalo *A(MPS)* for superior ao limite 0x8000, nada acontece e a rotina é terminada. Caso o limite for inferior a 0x8000, uma verificação é feita para saber se o valor do intervalo *A(MPS)* é menor do que o valor do sub-intervalo de *LPS* (*LSZ[ST[CX]]*). Caso isso seja verdade, uma troca condicional (*MPS/LPS*) é feita e o sub-intervalo de *LPS* será codificado em vez do sub-intervalo de *MPS*. A rotina calcula o valor da base do sub-intervalo de *LPS* e o tamanho do sub-intervalo de *LPS* (*A(LPS)*) usando as eqs. (5.3) e (5.4) mencionadas anteriormente. Após isso, a rotina determina o novo estado de estimativa de probabilidade do valor do *CX* particular relativo ao sub-intervalo de *A(MPS)*, utilizando a tabela de estimação de probabilidade [apêndice A], tanto para esse caso como para o caso em que o intervalo *A(MPS)* é maior do que *LSZ[ST[CX]]*. Finalmente, a rotina **MPSCOD** chama a rotina **RENORMCOD** para renormalizar os registradores *A* e *C*. A figura (5.7) mostra o fluxograma da rotina **MPSCOD**.

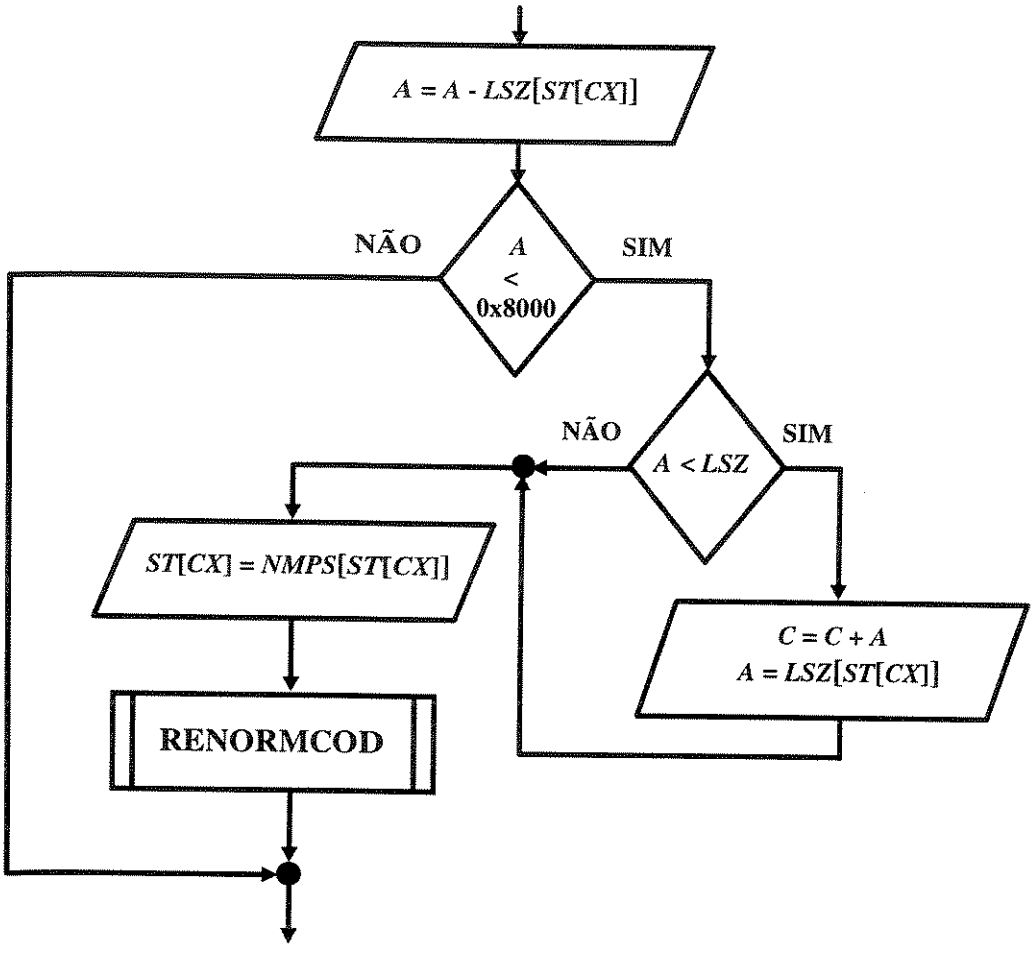


Fig. 5.7 - Fluxograma da rotina MPSCOD

5.3.2d) Rotina LPSCOD

A função básica dessa rotina é codificar qualquer *PIX* com menor ocorrência.

O início dessa rotina é parecido com a rotina **MPSCOD** na determinação do valor do sub-intervalo de *MPS*. Em seguida, compara-se o intervalo *A(MPS)* com o sub-intervalo de *LPS*. Caso o intervalo *A(MPS)* for maior, então, o intervalo *A(MPS)* é somado com sua base *C(MPS)* para produzir a base do sub-intervalo de *LPS* (*C(LPS)*) conforme a eq. (5.3). O intervalo *A(LPS)* é ajustado para o valor *LSZ[ST[CX]]* através da eq. (5.4). Após isso, tanto para esse caso como para o caso contrário (*MPS/LPS*), a rotina utiliza

a tabela de estimação de probabilidade [apêndice A] para verificar o valor do ***SWTCH[ST[CX]]*** do estado do valor do *CX* que está sendo codificado (*o SWTCH é um sinal de aviso indicando que o valor do PIX com maior ocorrência tem mudado. O PIX com maior ocorrência é o que está na variável MVAL[CX]*). Se *SWTCH*[*ST*[*CX*]]=1, a rotina converte o valor "0" ("1") da variável *MVAL*[*CX*] para o valor "1" ("0"), e ficaria o mesmo caso em que *SWTCH*[*ST*[*CX*]] = 0. A rotina continua para consultar novo estado de estimativa de probabilidade do valor do *CX* relativo ao sub-intervalo de *A*(*LPS*) utilizando-se a tabela de estimação de probabilidade [apêndice A]. No passo seguinte, a rotina **RENORMCOD** é chamada. A figura (5.8) mostra o fluxograma da rotina **LPSCOD**.

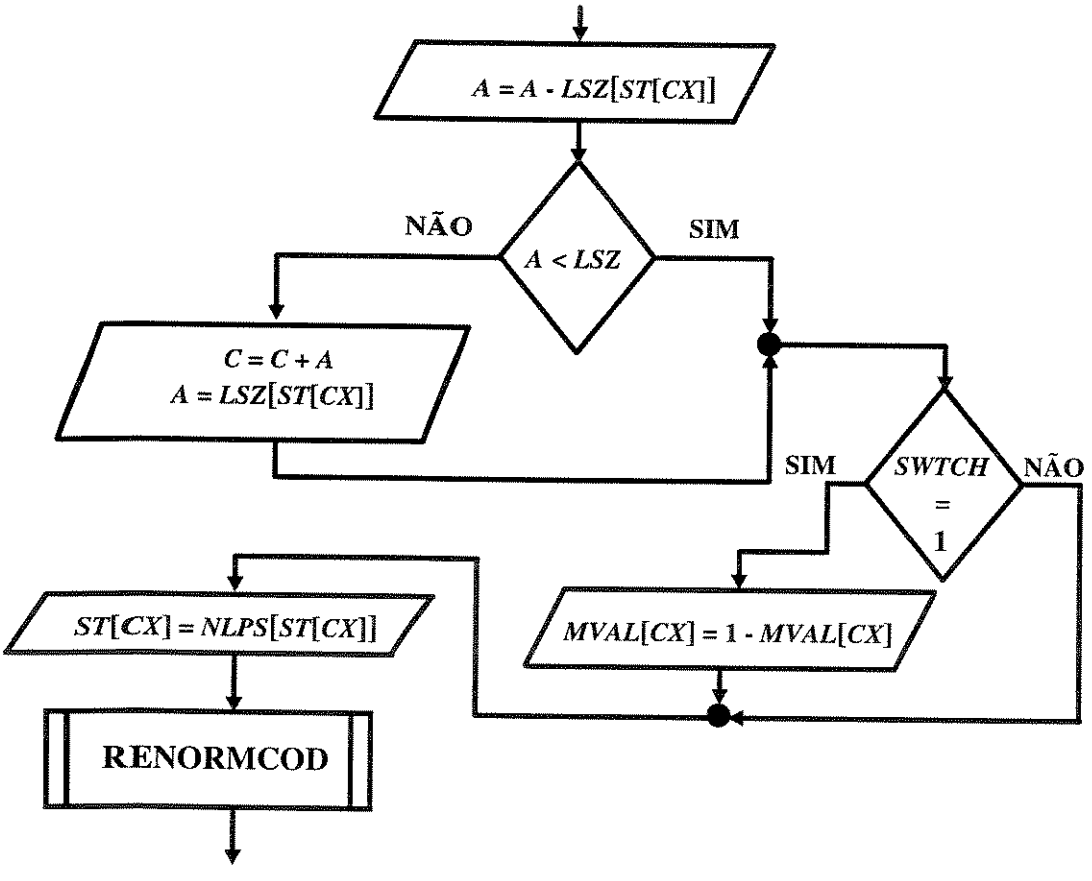


Fig. 5.8 - Fluxograma da rotina LPSCOD

5.3.2e) Rotina RENORMCOD

Essa rotina é chamada pelas rotinas **MPSCOD** e **LPSCOD** para renormalizar os registradores *A* e *C*. A rotina realiza um deslocamento do conteúdo dos registradores para a esquerda. Enquanto o valor do registrador *A* for inferior ao limite 0x8000, a rotina volta novamente para renormalizar os dois registradores. Durante esse tempo, o conteúdo do contador *CT* é subtraído de um valor igual a 1. Em seguida, a rotina faz questão de testar o conteúdo de *CT*. Se for igual 0, a rotina **BYTEOUT** é chamada para transmitir ou armazenar 1 *byte* de palavra código (*PCT*). O fluxograma da rotina **RENORMCOD** está mostrado na figura (5.9).

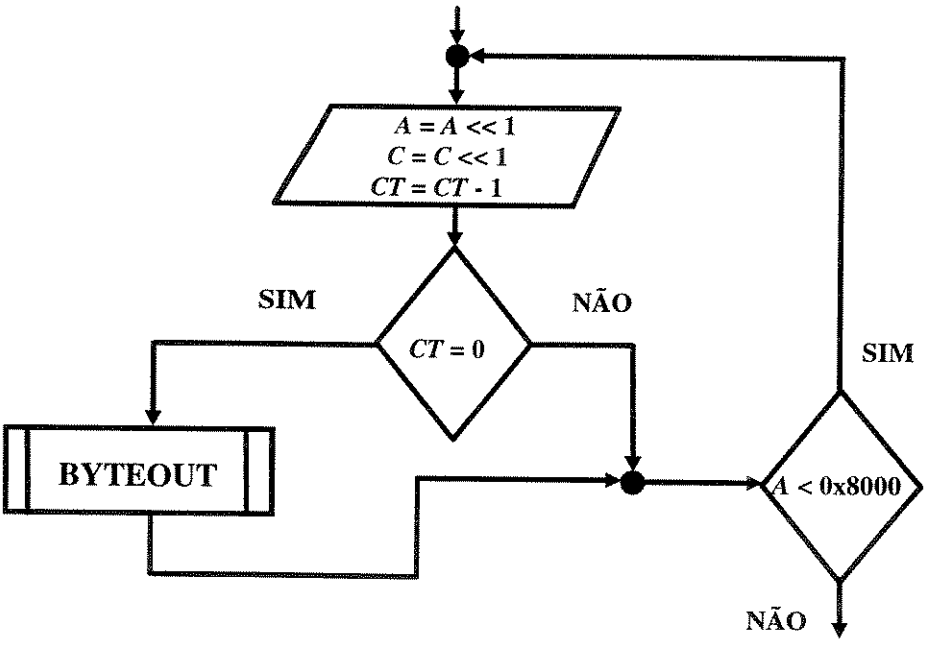


Fig. 5.9 - Fluxograma da rotina RENORMCOD

5.3.2f) Rotina BYTEOUT

Essa rotina é chamada pela rotina RENORMCOD. A rotina tem uma variável temporária ***TEMP*** para armazenar 1 *byte* no lado **MSB** do registrador *C* (os *bits* de p, conforme tabela (5.1)) mais 1 *bit* de *carry* (o *bit* de c) *para fazer teste de carry-over.*

O processo começa com o deslocamento do conteúdo do registrador *C* 19 vezes para a direita. O resultado é transferido para a variável *TEMP*. Em seguida, a rotina testa o conteúdo de *TEMP*, ou seja, testa a ocorrência do *carry-over*.
Existem três casos para teste:

1. Se o conteúdo for maior do que o valor 0xff, isso significa que tem *carry-over*. Esse *carry-over* deve ser somado com a recente tentativa de dados de saída que está no *BUF*. Após isso, a rotina transmite ou armazena o conteúdo do *BUF*. Em seguida, a rotina transmite o valor 0x00 *SC* vezes, em outras palavras, transmite *SC bytes* de valor 0x00. Logo, zera-se o conteúdo do contador *SC*. No último passo, uma nova tentativa de dados de saída é determinada colocando-se o conteúdo de *TEMP* no *BUF* sem nenhum *carry* ;

2. Se o conteúdo for menor do que o valor 0xff, significa que o problema do *carry-over* está resolvido. A rotina transmite ou armazena o conteúdo de *BUF*. Logo em seguida, transmite *SC* vezes o valor 0xff e o conteúdo do contador *SC* é zerado. O novo valor é armazenado no *BUF* e é igual ao valor de *TEMP* ;

3. Se o conteúdo for igual ao valor 0xff, não se transmite nenhum dado de saída e o conteúdo do contador *SC* é incrementado de um valor igual a 1.

Finalmente, depois do procedimento de teste a rotina ajusta o valor do registrador *C*. O conteúdo será os *bits* fracionais do valor de base de intervalo *A* (os *bits* de b, conforme tabela (5.1)) mais os *bits* de *carry-over* (os *bits* de s). O conteúdo do contador *CT* é ajustado para o valor 8. A figura (5.10) mostra o fluxograma da rotina **BYTEOUT**.

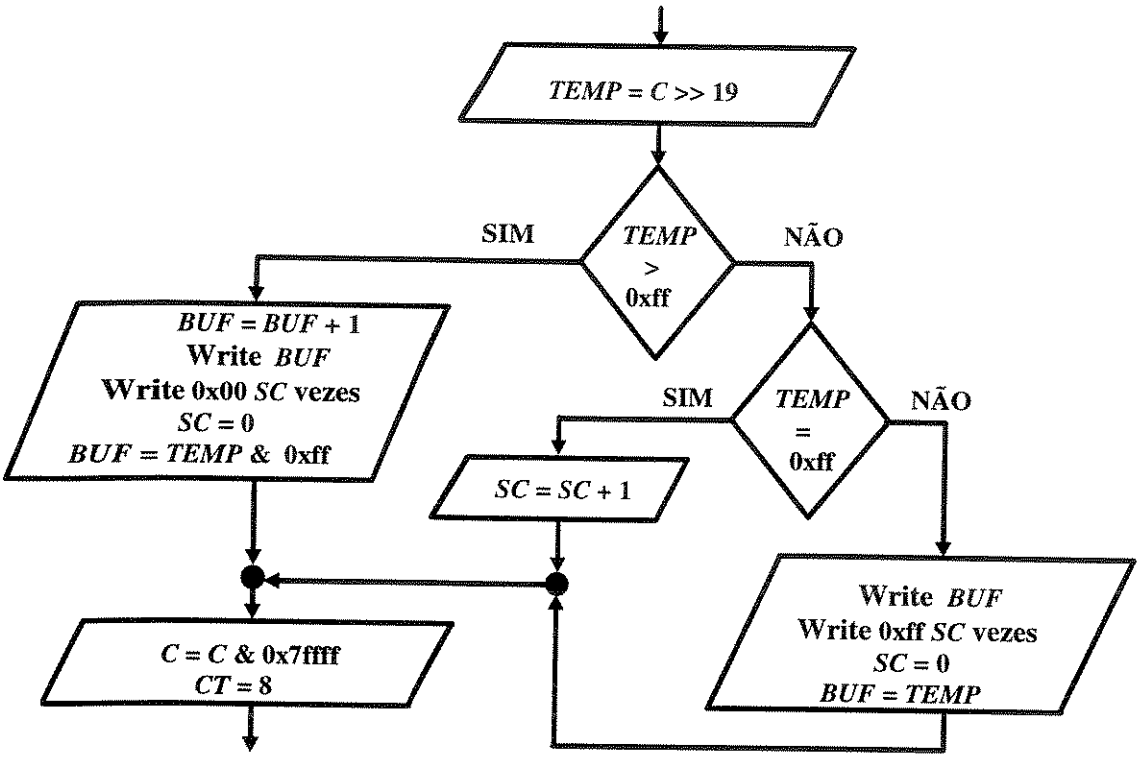


Fig. 5.10 - Fluxograma da rotina BYTEOUT

5.3.2g) Rotina FIMC

Essa rotina internamente chama duas rotinas. A primeira é a **ZERARBITS** e a segunda é a **ESCREVFIM**. O fluxograma dessa rotina está mostrado na figura (5.11).

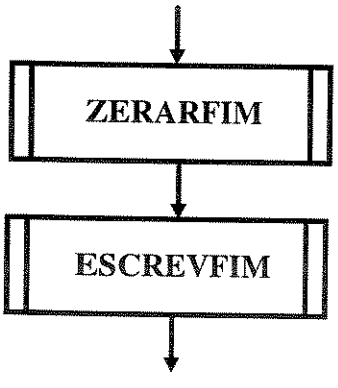


Fig. 5.11 - Fluxograma da rotina FIMC

5.3.2h) Rotina ZERARBITS

O conteúdo do registrador *C* é ajustado para um valor dentro do intervalo [*C* , *C* + *A* - 1] com o maior número de *bits* "0" possíveis [7]. A figura (5.12) mostra o fluxograma dessa rotina.

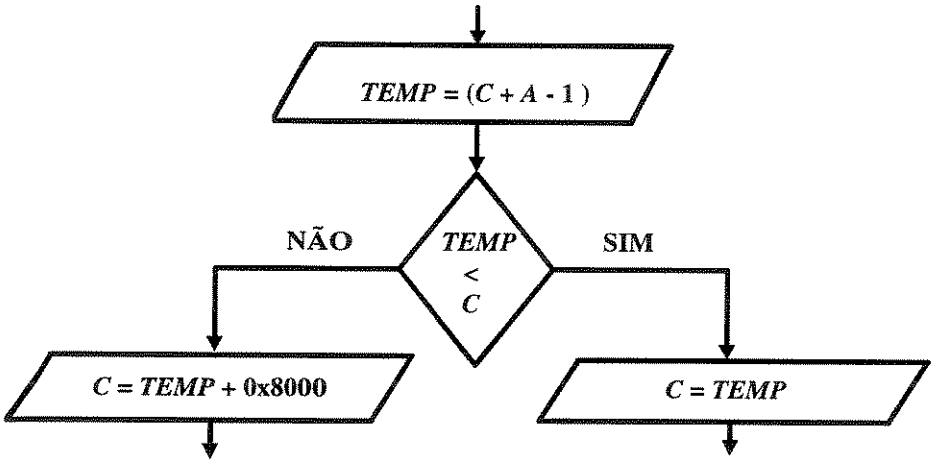


Fig. 5.12 - Fluxograma da rotina ZERARBITS

5.3.2k) Rotina ESCREVFIM

A função dessa rotina é resolver o problema de *carry*-over. É semelhante a rotina **BYTEOUT**. A rotina testa o *BUF*. Se esse não for igual a 0, passa-se para o teste do *carry-over*. Se ocorrer *carry-over*, envia-se o conteúdo do *BUF* somado de um valor igual a 1 para a saída e em seguida, o valor 0x00 é enviado *SC* vezes. Caso contrário, a rotina envia o conteúdo do *BUF* seguido pelo valor 0xff *SC* vezes. Finalmente, 2 *bytes* do registrador *C* são transmitidos sendo que isso serve também se o conteúdo do *BUF* for igual a 0. A figura (5.13) mostra o fluxograma da rotina **ESCREVFIM**.

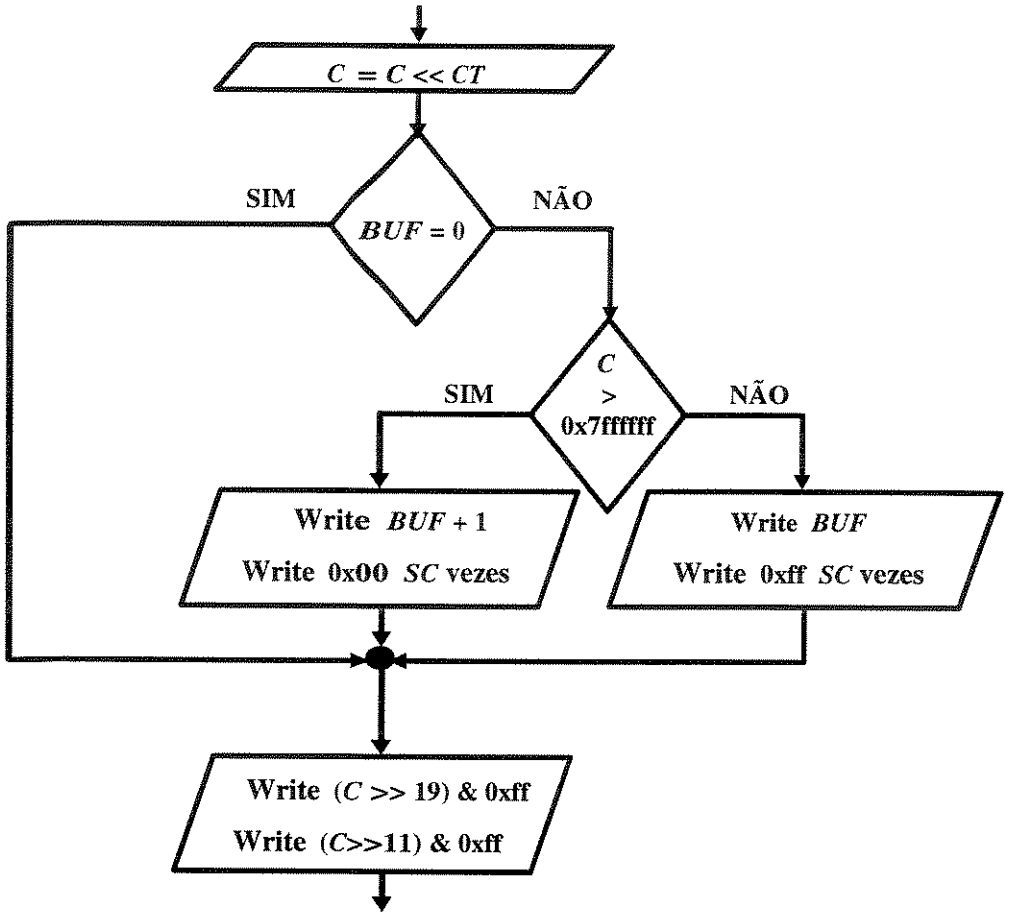


Fig. 5.13 - Fluxograma da rotina ESCREVFIM

5.4 Processo de Decodificação Aritmética Binária Adaptativa (*ABAD*)

Como foi dito no processo de codificação, considerando-se o caso em que não se usa o modo progressivo, o procedimento do **decodificador aritmético adaptativo** (***DAA***) utiliza somente dois fluxos de entrada: **o *pixel* codificado *PCT* e o contexto *CX***. A saída produz somente um fluxo de ***pixels* recuperados *PIX***. A figura (5.14) mostra a entrada e a saída do sub-bloco de *DAA*.

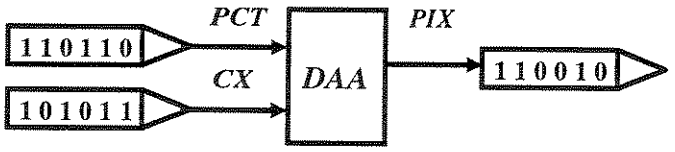


Fig. 5.14 - Entrada e saída do sub-bloco de DAA

O tratamento será com uma única camada de resolução sem referência às outras camadas de resolução. Quando o procedimento de *CAA* codifica um intervalo A, o procedimento de *DAA* faz questão de determinar qual o sub-intervalo que foi codificado, em outras palavras, qual o sub-intervalo que está sendo apontado pelo fluxo de *pixel* codificado *PCT*. Isso indica que o processo de decodificação também é feito de maneira recursiva.

Durante o processo de decodificação, o procedimento de *DAA* é subtrair qualquer sub-intervalo que o procedimento de *CAA* somou ao fluxo *PCT* (registrador C). Portanto, o fluxo *PCT* no decodificador é um ponteiro apontado para um número real dentro do intervalo A , relativo a sua base.

5.4.1 Rotinas e Fluxogramas do *DAA*

O procedimento do *DAA* é *executado para cada pixel do fluxo CX e produz na saída o fluxo PIX dos pixels recuperados*. O *DAA* internamente contém duas rotinas: a **INITD** para inicializar as variáveis e registradores utilizados pelo *DAA*; e a rotina **DECODIF** para decodificar o fluxo *PCT*, ou seja, para recuperar os *pixels* originais do fluxo *PIX*. O fluxograma do *DAA* está mostrado na figura (5.15). O *DAA* usa somente o registrador C_H para fazer comparação. 1 *byte* do fluxo *PCT* é inserido no **MSB** do registrador C_L sempre que necessário para dar continuidade ao processo de decodificação.

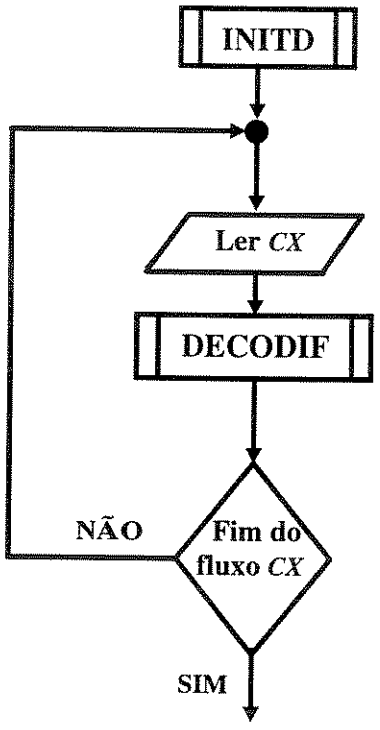


Fig. 5.15 - Fluxograma do DAA

5.4.1a) Rotina INITD

A função dessa rotina é a de inicialização. Para todos os valores possíveis do *CX*, os estados de estimação de probabilidade são inicializados com o valor "0" e a variável *MVAL*[*CX*] com o valor "0"; o registrador *A* com o valor "0x10000"; o registrador *C* com o valor "0". Essa rotina utiliza a rotina **LERBYTE** para ler 3 *bytes* do fluxo *PCT* e inicializa o contador *CT* com o valor 8. A figura (5.16) mostra o fluxograma da rotina **INITD**.

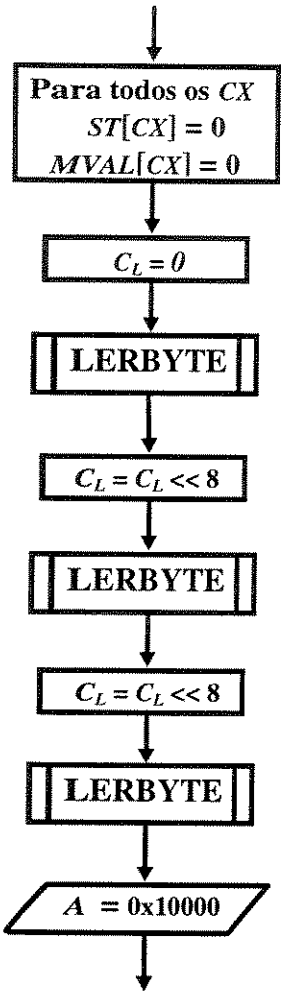


Fig. 5.16 - Fluxograma da rotina INITD

5.4.1b) Rotina LERBYTE

O fluxograma dessa rotina está mostrado na figura (5.17). A rotina testa se ainda existem *pixels* do fluxo *PCT*. Se não existirem mais *pixels*, o conteúdo do *BUF* é ajustado para o valor "0". Caso contrário, 1 *byte* do fluxo *PCT* é inserido nos oito *bits* superiores do registrador C_L. Em seguida, o conteúdo do contador *CT* é inicializado com o valor 8.

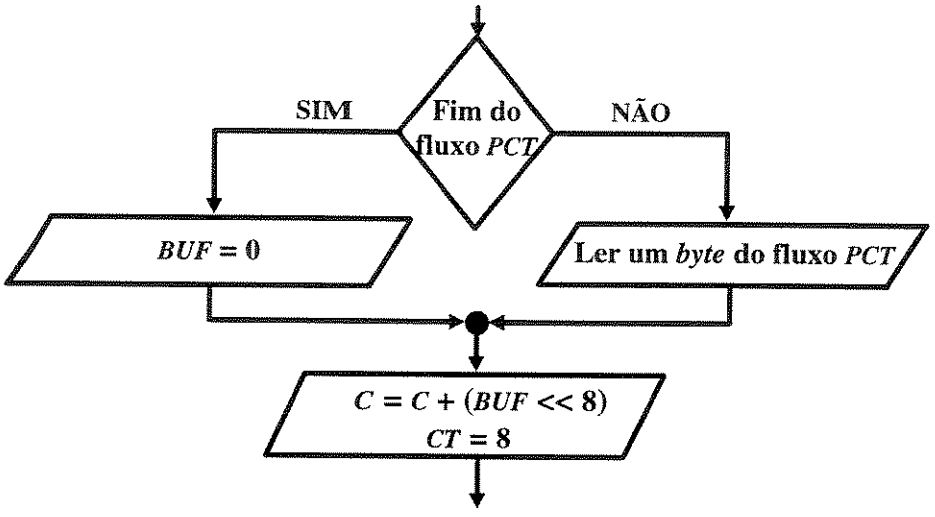


Fig. 5.17 - Fluxograma da rotina LERBYTE

5.4.1c) Rotina DECODIF

A rotina começa determinando o intervalo $A(MPS)$ para se fazer a comparação com o conteúdo do registrador C_H. Se o intervalo $A(MPS)$ for maior do que o conteúdo do registrador C_H, um teste do intervalo $A(MPS)$ é feito; se o conteúdo for inferior ao limite "0x8000", o *pixel* recuperado é igual ao *bit* da variável $MVAL[CX]$. Caso contrário, a rotina **MPSDECOD** é chamada. Se o intervalo for menor ou igual ao conteúdo do registrador C_H , a rotina **LPSDECOD** é chamada. Toda vez que a rotina **MPSDECOD** ou **LPSDECOD** for solicitada, a rotina **RENORMDEC** é chamada. A figura (5.18) mostra o fluxograma da rotina **DECODIF**.

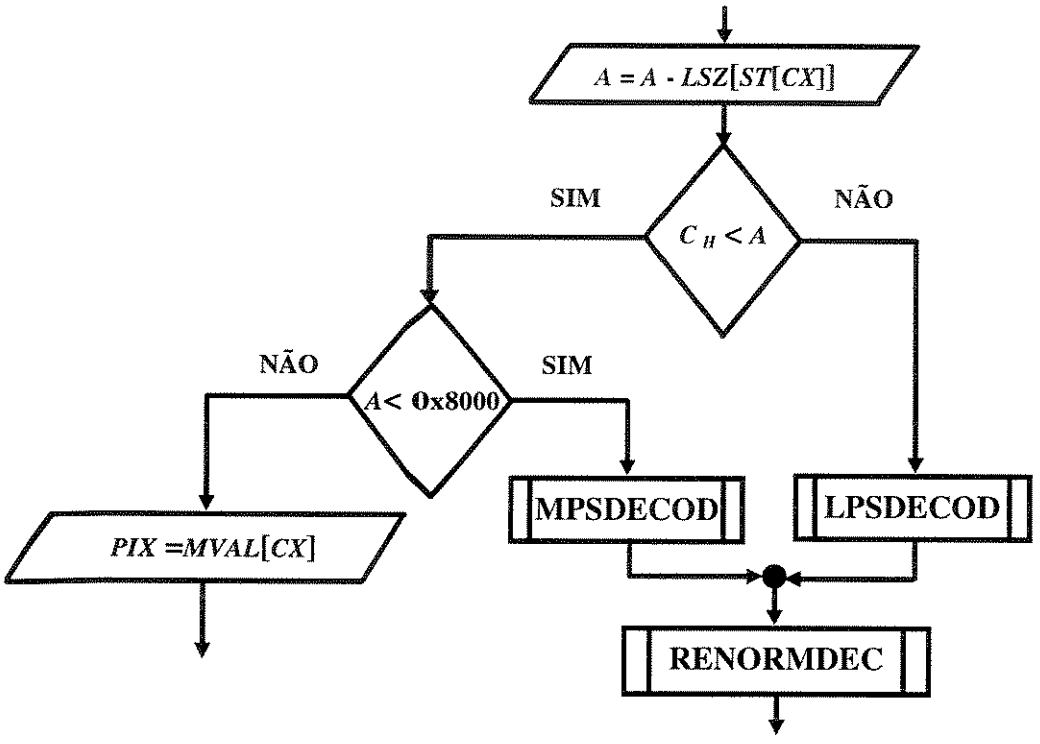


Fig. 5.18 - Fluxograma da rotina DECODIF

5.4.1d) Rotina MPSDECOD

A figura (5.19) mostra o fluxograma dessa rotina. A função básica é decodificar o *PIX* com maior ocorrência. Se o sub-intervalo de *A(MPS)* for menor do que o sub-intervalo de *A(LPS)*, isso significa que o sub-intervalo de *A(LPS)* será decodificado, ou seja, uma troca condicional (*MPS/LPS*) ocorrerá. O *PIX* recuperado é o conteúdo da variável *MVAL*[*CX*] convertido. O *SWTCH*[*ST*[*CX*]] do estado do valor do *CX* é testado. Nesse caso, se *SWTCH*[*ST*[*CX*]] = 1, o valor "0"("1") da variável *MVAL*[*CX*] é convertido para o valor "1"("0") e ficará o mesmo caso *SWTCH*[*ST*[*CX*]] = 0. A rotina consulta o novo estado de estimativa de probabilidade do valor do *CX* relativo ao sub-intervalo de *A(LPS)* da tabela de estimação de probabilidade [apêndice A]. Caso o sub-intervalo de *A(MPS)* seja maior do que o sub-intervalo de *A(LPS)* então o sub-intervalo de *A(MPS)* será decodificado, a rotina determina o *PIX* recuperado e consulta-se o novo estado de

estimativa de probabilidade do valor do *CX* relativo ao sub-intervalo de *A(MPS)* consultando-se a tabela de estimação de probabilidade [apêndice A].

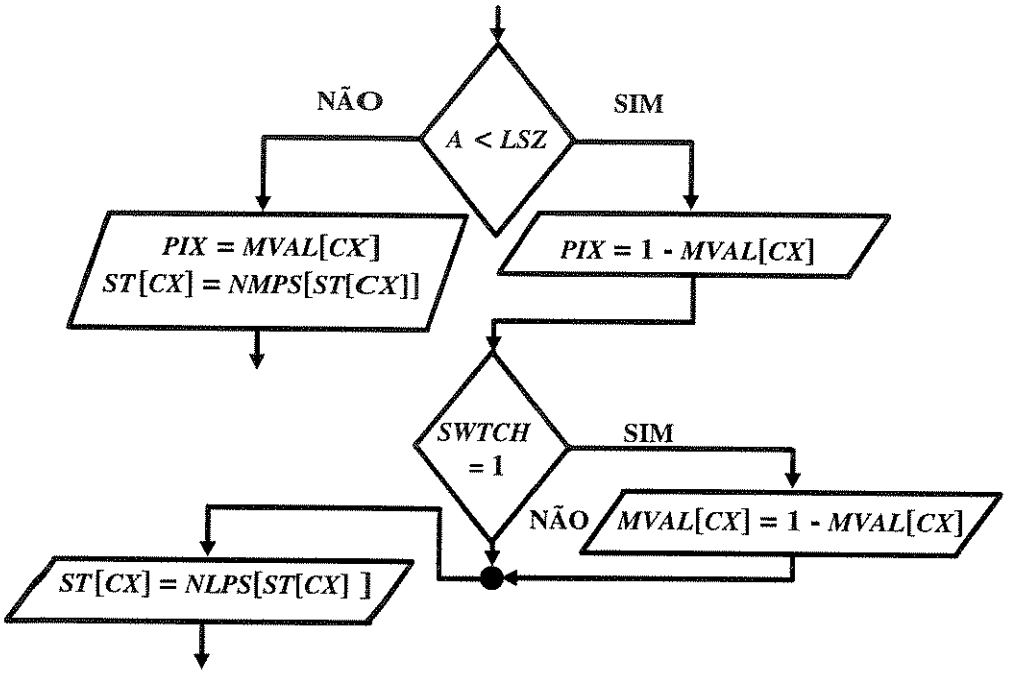


Fig. 5.19 - Fluxograma da rotina MPSDECOD

5.4.1e) Rotina LPSDECOD

Basicamente, essa rotina decodifica o *PIX* com menor ocorrência. Se por acaso o sub-intervalo de *A(MPS)* for menor do que o sub-intervalo de *A(LPS)*, uma troca condicional (*MPS/LPS*) ocorre e o sub-intervalo de *A(MPS)* é decodificado. O conteúdo do registrador C_H é subtraído pelo conteúdo do registrador *A* que foi somado pelo *CAA*. O novo intervalo de *A* é determinado. O *PIX* recuperado será o conteúdo da variável *MVAL[CX]*. Em seguida, o novo estado de estimativa de probabilidade relativo ao sub-intervalo de *A(MPS)* é consultado da tabela de estimação de probabilidade para o *CX* que está sendo decodificado. Caso o sub-intervalo de *A(MPS)* for maior do que o sub-intervalo de *A(LPS)*, o conteúdo do registrador C_H é subtraído pelo conteúdo do registrador *A*. O novo conteúdo do registrador *A* é igual ao sub-intervalo de *A(LPS)*

(*LSZ*[*ST*[*CX*]]) e o *PIX* recuperado é calculado. O *bit* de *SWTCH*[*ST*[*CX*]] do estado do valor do *CX* é testado, se *SWTCH*[*ST*[*CX*]] = 1 e o conteúdo da variável *MVAL*[*CX*] é convertido. Após isso, a rotina consulta na tabela de estimação de probabilidade o novo estado de estimativa de probabilidade do valor do *CX* relativo ao sub-intervalo de *A*(*LPS*) e o mesmo é feito caso o *SWTCH*[*ST*[*CX*]] = 0. A figura (5.20) mostra o fluxograma dessa rotina.

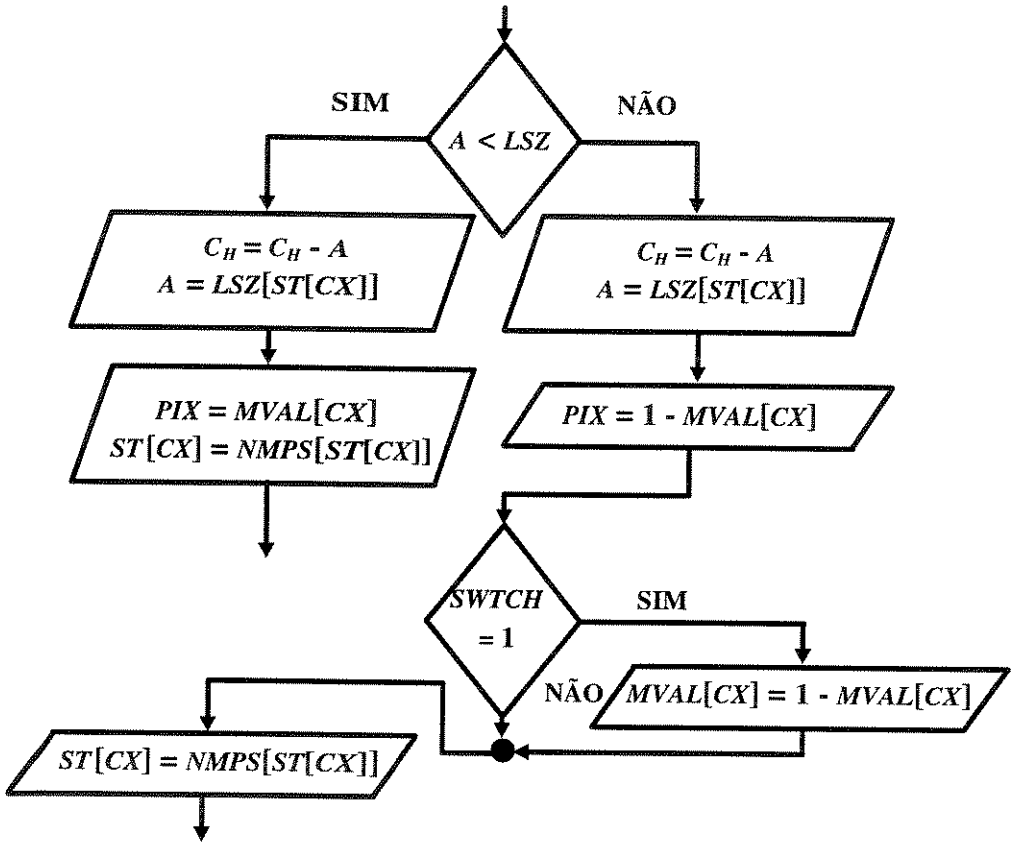


Fig. 5.20 - Fluxograma da rotina LPSDECOD

5.4.1f) Rotina RENORMDEC

Essa rotina é chamada pela rotina **DECODIF** toda vez que o *PIX* é recuperado (decodificado). O contador *CT* mantém o número de *bits* compactados no registrador C_L. Quando o conteúdo do contador *CT* for igual a 0, 1 *byte* do fluxo *PCT* é inserido no lado

superior (**MSB**) do registrador C_L . Durante o processo ambos os registradores A e C (C_H + C_L) são deslocados de 1 *bit* para a esquerda até que o conteúdo do registrador A seja inferior ao limite "0x8000". Em seguida um novo teste de contador CT é feito e quando o conteúdo for igual a 0, 1 *byte* do fluxo PCT é inserido no lado superior do registrador C_L . O fluxograma dessa rotina está mostrado na figura (5.21).

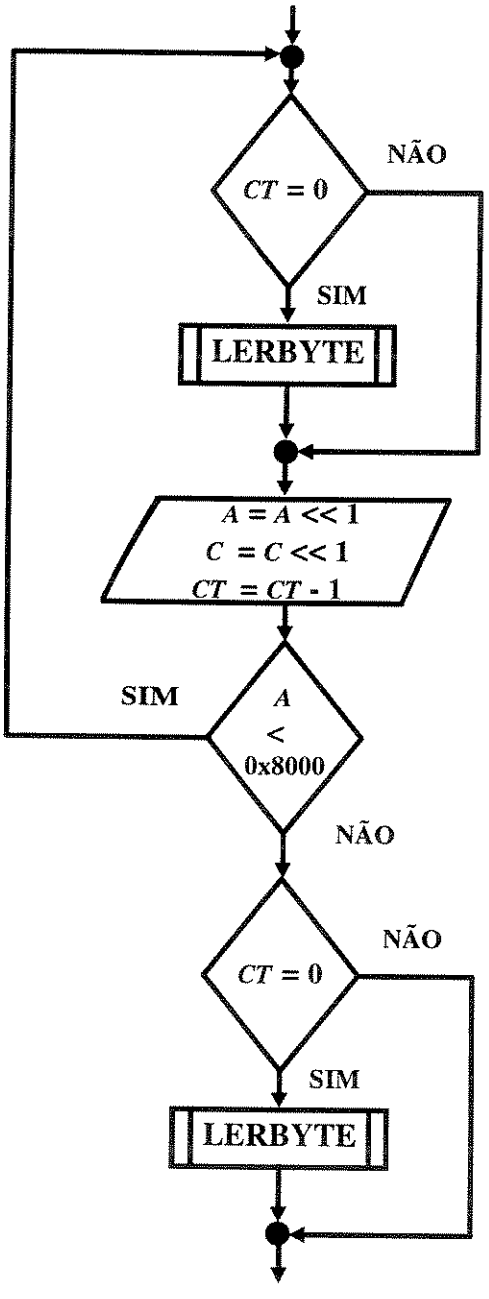


Fig. 5.21 - Fluxograma da rotina RENORMDEC

Capítulo 6

SIMULAÇÕES E RESULTADOS

6.1 Introdução

A codificação aritmética binária adaptativa (*ABAC*), que foi utilizada neste trabalho, é implementada para imagens paradas (estáticas) de 2 *tons*. Essa *ABAC* é capaz de proporcionar boas taxas de compressão para imagens de 2 *tons*, uma porcentagem de compressão alta e permite o modo de operação sem perdas (a imagem recuperada é exata).

Geralmente, a avaliação do desempenho de um sistema de comunicações é realizada através do uso de aparelhos de testes que medem os parâmetros que caracterizam tal sistema. No caso de desenvolvimento de projetos, tais medidas objetivas são essenciais. Também é possível obter uma avaliação do desempenho de um sistema através da simulação em computadores que permitem testar ou avaliar o desempenho do sistema através de uso das imagens de teste, aproveitando-se a rapidez e a capacidade associados aos computadores.

A vantagem das simulações em relação à implementação física é evidente quando se tem um número grande de opções que compartilham soluções para a construção do sistema.

Neste capítulo, apresentamos os recursos usados, os tipos das imagens de teste, seus formatos e a avaliação do desempenho do algoritmo de *ABAC*.

6.2 Plataforma de Trabalho

Como plataforma de trabalho, a simulação da *ABAC* é implementada com a ajuda de uma **estação de trabalho** (***SUN SPARCstation* 20**) com o **sistema operacional** (***SunOS 5.5 Generic***). Aproveita-se a facilidade do ambiente do ***Open Windows*** versão 3.5 / ***Solaris*** versão 2.5, sendo que os programas em software são desenvolvidos em **linguagem C** (*ANSI gcc* - versão 2.7.0).

Através dessa plataforma foi gerado o simulador ***JBIG_CODE*** para realizar as operações de codificação/decodificação aritmética binária adaptativa.

O simulador *JBIG_CODE* possui uma janela principal (figura (6.1)) com as opções de codificação e decodificação.

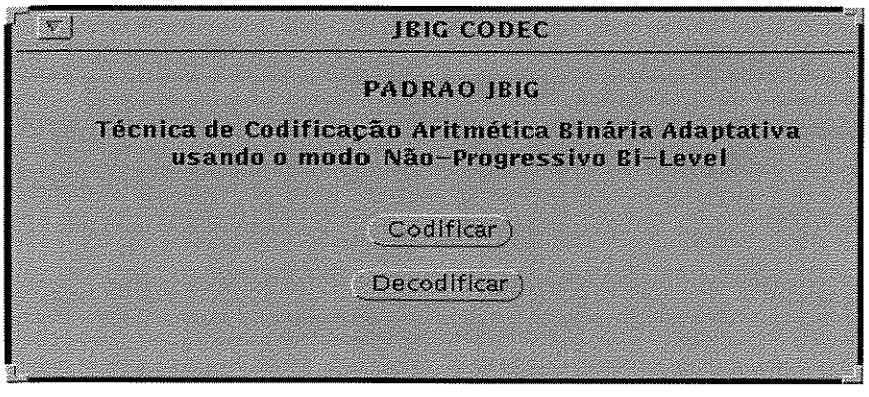


Fig. 6.1 - Janela principal de simulador JBIG_CODE

A opção de codificação (figura (6.2)) permite que o usuário forneça dados sobre os arquivos da:

⇒ Imagem Fonte "de teste" (formato *ASCII* tipo *PBM*);

⇒ Contexto (extensão *CX*);

⇒ Imagem codificada (extensão *PCT*).

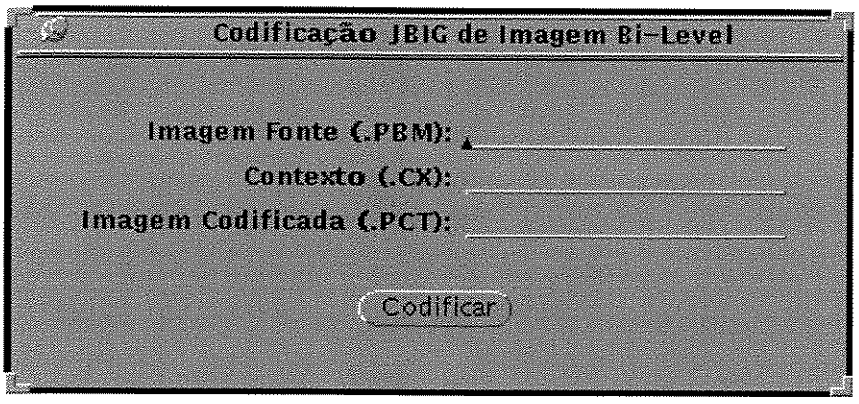


Fig. 6.2 - A janela da opção de codificação

A opção de decodificação abre uma janela como está mostrada na figura (6.3) com os campos para:

⇒ Imagem codificada (extensão *PCT*);

⇒ Contexto (extensão *CX*);

⇒ Imagem recuperada (formato *ASCII* tipo *PBM*).

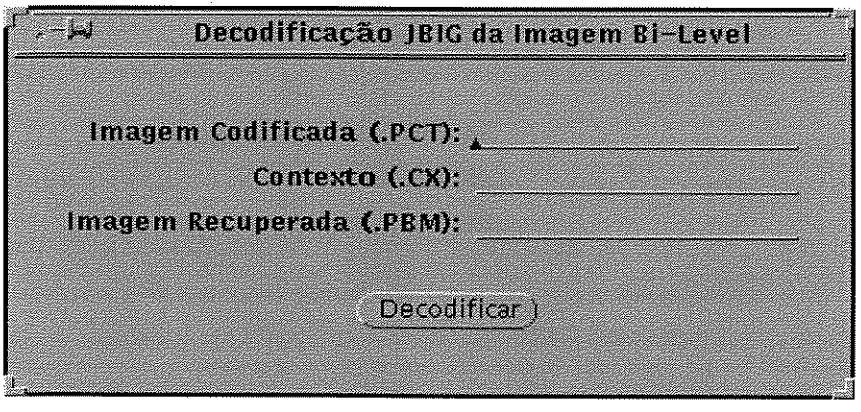


Fig. 6.3 - A janela da opção de decodificação

O fluxograma da figura (6.4) mostra de uma forma geral os vários procedimentos para o uso do simulador.

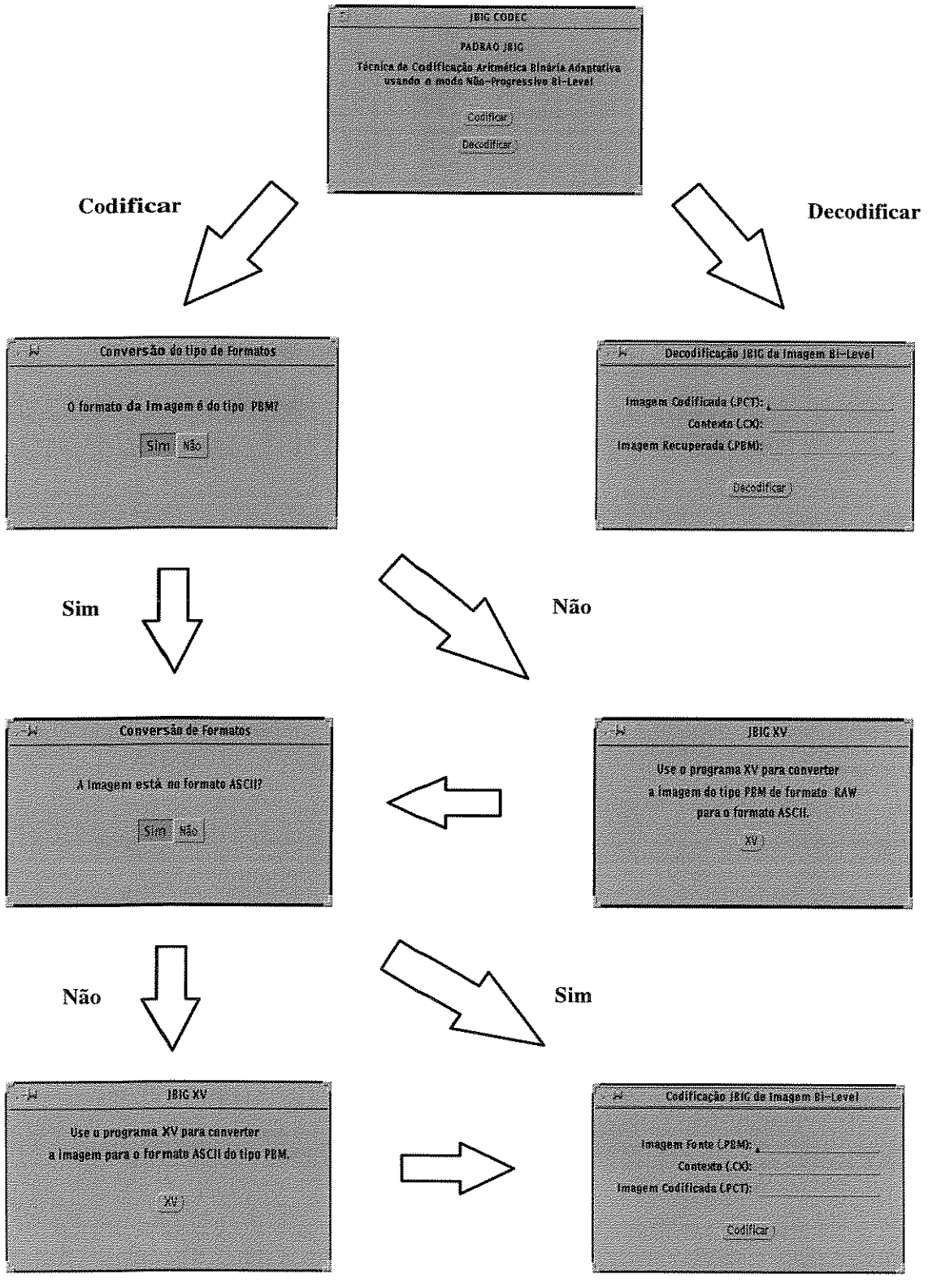
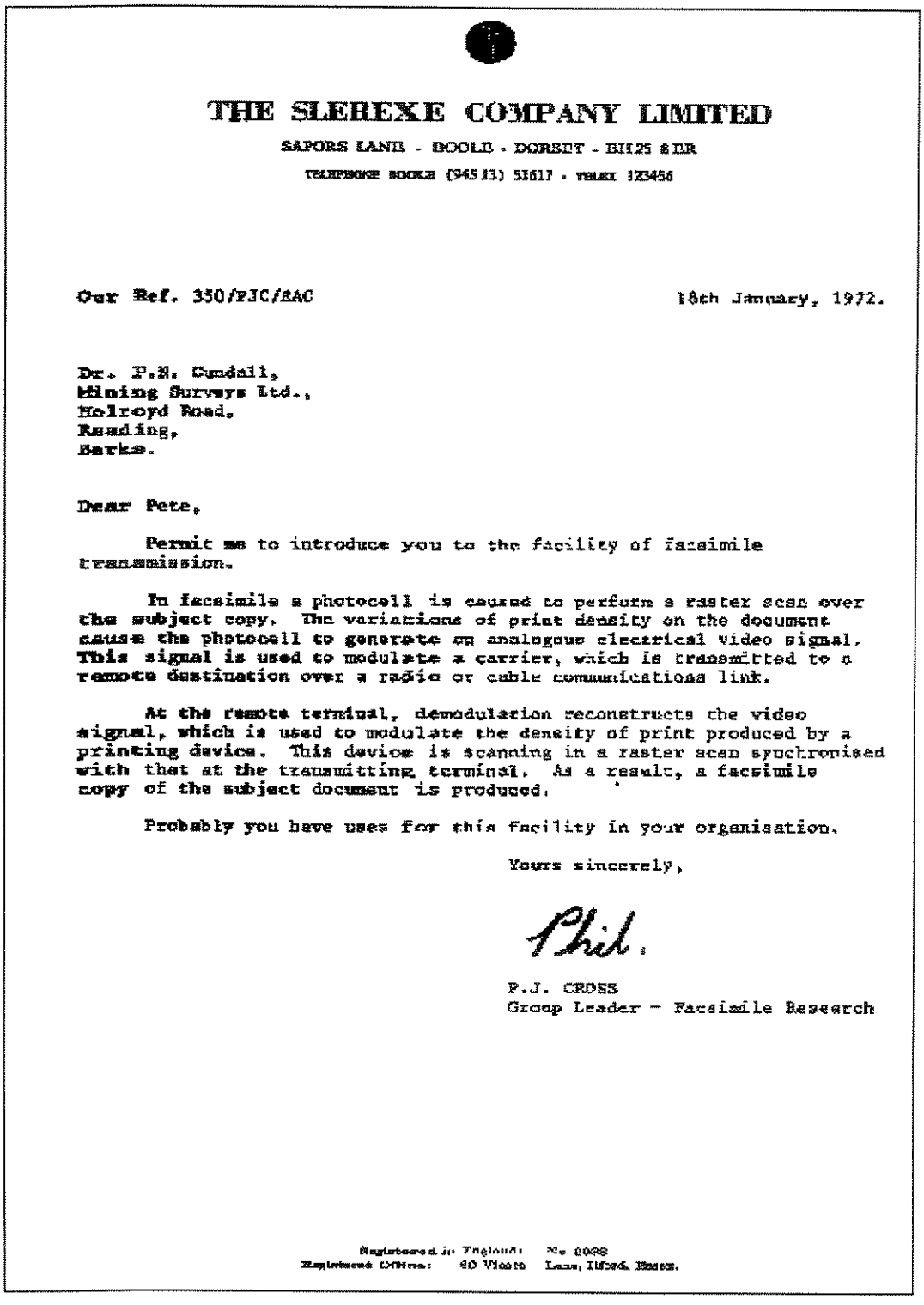


Fig. 6.4 - Fluxograma de procedimentos da codificação/decodificação

6.3 Imagens de Teste

As imagens de teste são usadas como imagens fonte. Nessa simulação, dois tipos de imagens de teste foram utilizados pelo algoritmo de *ABAC*:

- **Primeiro Tipo**: Imagens padronizadas pelo ***CCITT***. São 8 imagens que possuem características diferentes que permitem avaliar e examinar a habilidade do sistema (codificador aritmético adaptativo - *CAA*) de forma geral. De acordo com os padrões para fac-símile digital do *CCITT*, essas imagens têm 1728 elementos de imagem (*pixels*) ou pontos a serem lidos pelo ***Scanner*** ao longo da largura de um documento de 21cm de largura, e 2376 *pixels* ao longo da altura de um documento de 29,7cm de altura (imagem com formato 1728x2376 *pels* - **uma folha de tipo A_4**). As figuras (6.5) a (6.12) mostram as 8 imagens de padrão *CCITT*.



THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

TELEPHONE BOOLE (945 13) 51617 - TELEX 123456

Our Ref. 350/PJC/RAC 18th January, 1972.

Dr. P.H. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Registered in England No. 0088
Registered Office: 60 Victoria Lane, Ilford, Essex.

Fig. 6.5 - Imagem de CCITT1.PBM

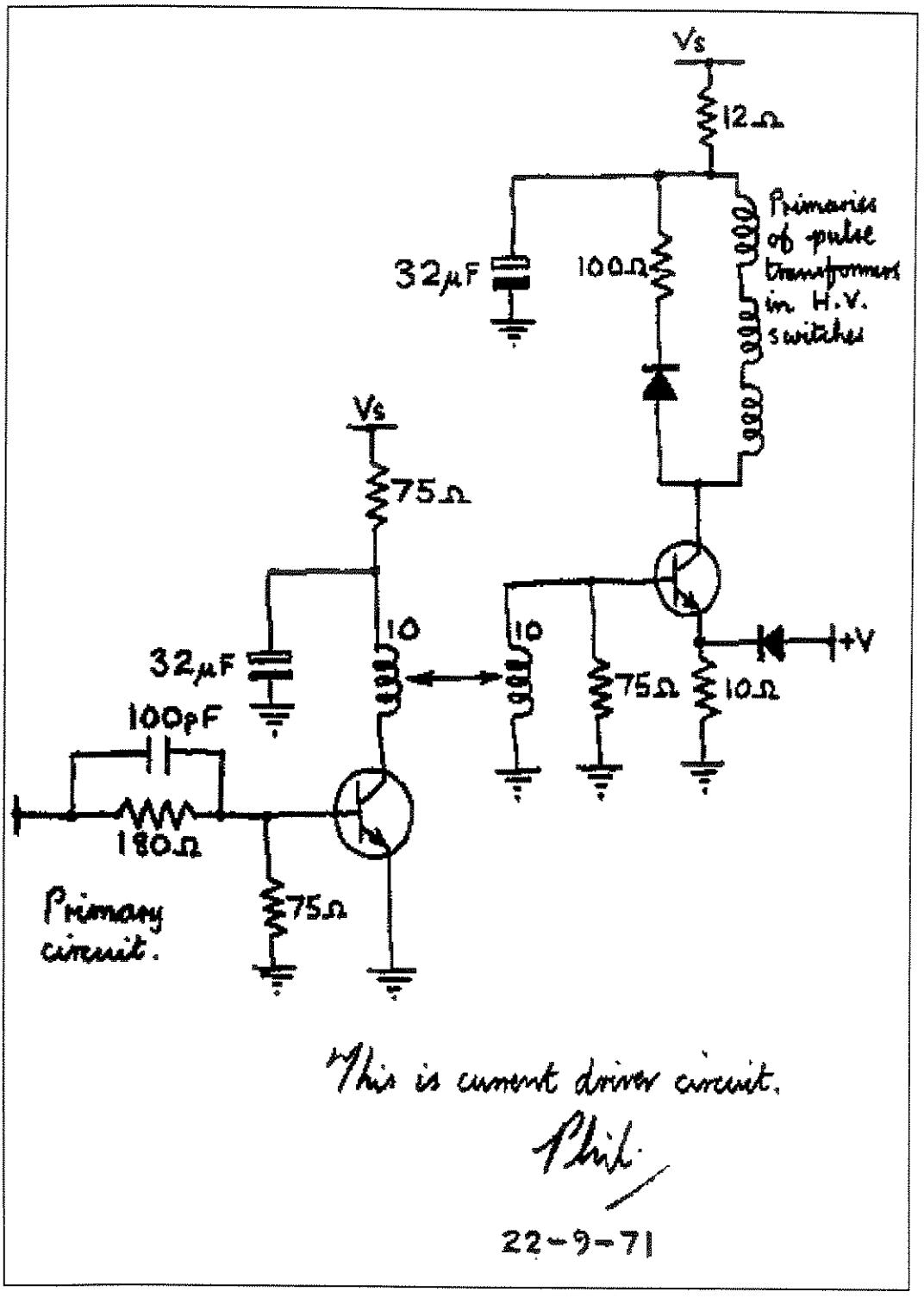
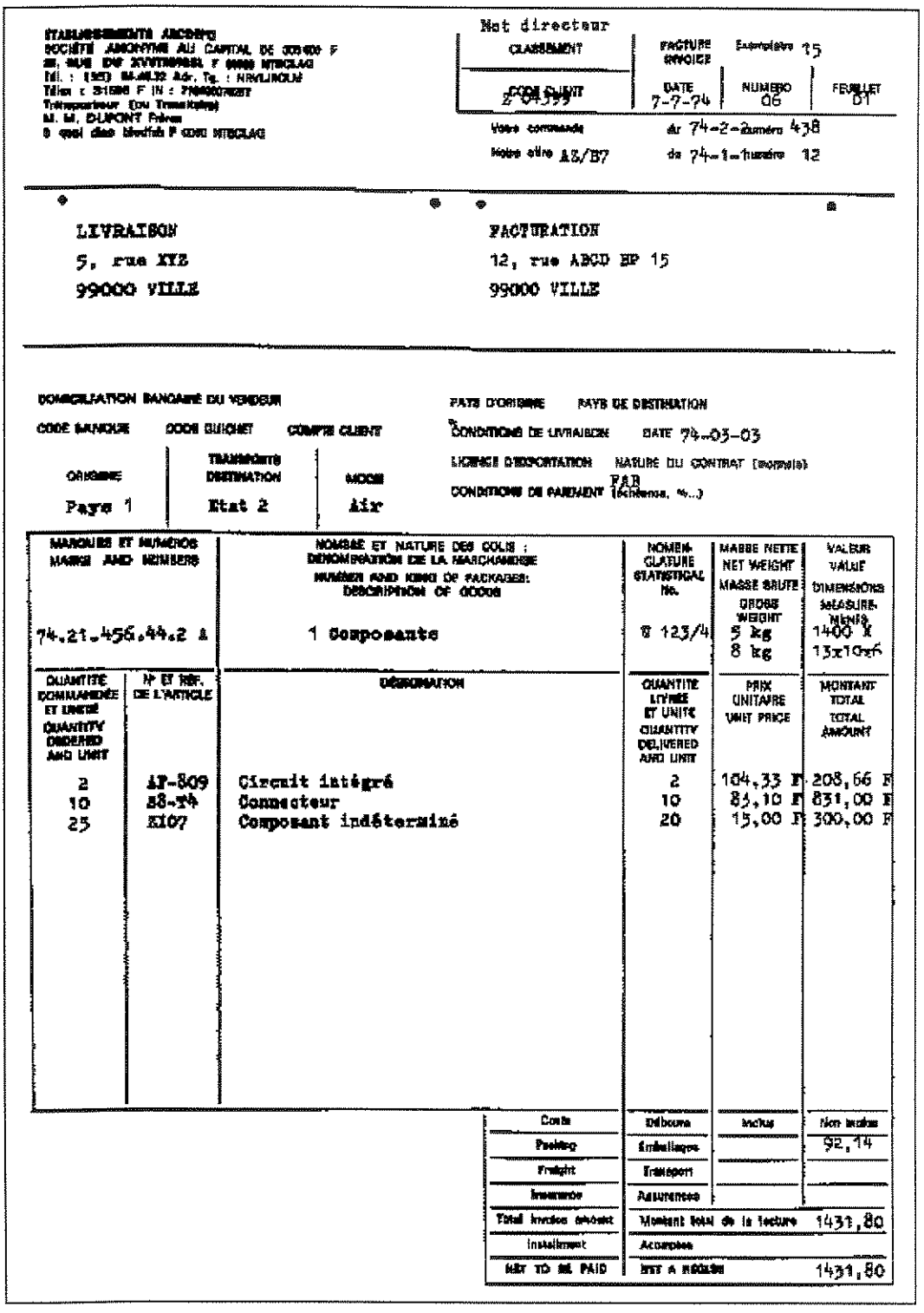


Fig. 6.6 - Imagem de CCITT2.PBM



Mot directeur

CLASSEMENT | FACTURE INVOICE Exemplaire 15 | CODE CLIENT 2 04399 | DATE 7-7-74 | NUMERO 06 | FEUILLET 01

Votre commande du 74-2-numéro 438

Notre offre AZ/B7 du 74-1-numéro 12

LIVRAISON
5, rue XYZ
99000 VILLE

FACTURATION
12, rue ABCD BP 15
99000 VILLE

DOMICILIATION BANCAIRE DU VENDEUR
CODE BANQUE | CODE GUICHET | COMPTE CLIENT

PAYS D'ORIGINE | PAYS DE DESTINATION
CONDITIONS DE LIVRAISON | DATE 74-03-03
LICENCE D'EXPORTATION | NATURE DU CONTRAT (facultatif)
CONDITIONS DE PAIEMENT (échéance, %...) PAR

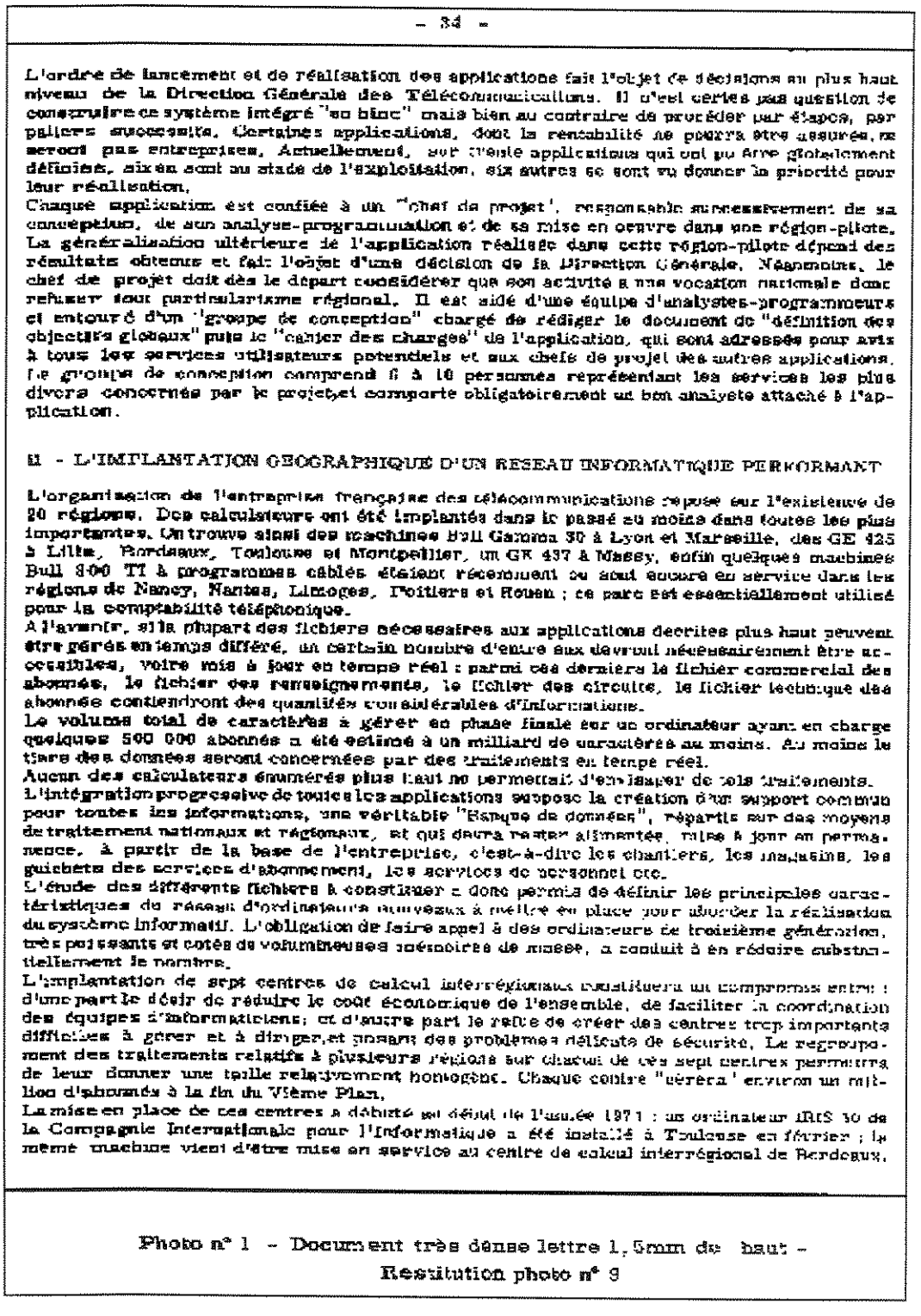
ORIGINE	TRANSPORTS DESTINATION	MODE
Pays 1	Etat 2	air

MARQUES ET NUMEROS MARKS AND NUMBERS	NOMBRE ET NATURE DES COLIS ; DENOMINATION DE LA MARCHANDISE NUMBER AND KIND OF PACKAGES; DESCRIPTION OF GOODS	NOMENCLATURE STATISTIQUE No.	MASSE NETTE NET WEIGHT	MASSE BRUTE GROSS WEIGHT	VALEUR VALUE	DIMENSIONS MEASUREMENTS
74.21.456.44.2 A	1 Composante	8 123/4	5 kg	8 kg	1400 X	13x10cm

QUANTITE COMMANDEE ET UNITE QUANTITY ORDERED AND UNIT	N° ET REF. DE L'ARTICLE	DESIGNATION	QUANTITE LIVREE ET UNITE QUANTITY DELIVERED AND UNIT	PRIX UNITAIRE UNIT PRICE	MONTANT TOTAL TOTAL AMOUNT
2	AP-809	Circuit intégré	2	104,33 F	208,66 F
10	58-74	Connecteur	10	83,10 F	831,00 F
25	X107	Composant indéterminé	20	15,00 F	300,00 F

	Débours	Inclus	Non inclus
Coste			
Packing	Emballages		92,14
Freight	Transport		
Insurance	Assurances		
Total invoice amount	Montant total de la facture		1431,80
Instalment	Acomptes		
NET TO BE PAID	NET A REGLER		1431,80

Fig. 6.7 - Imagem de CCITT3.PBM



– 34 –

L'ordre de lancement et de réalisation des applications fait l'objet de décisions au plus haut niveau de la Direction Générale des Télécommunications. Il n'est certes pas question de construire ce système intégré "au blanc" mais bien au contraire de procéder par étapes, par paliers successifs. Certaines applications, dont la rentabilité ne pourra être assurée, ne seront pas entreprises. Actuellement, sur trente applications qui ont pu être globalement définies, six en sont au stade de l'exploitation, six autres se sont vu donner la priorité pour leur réalisation.
Chaque application est confiée à un "chef de projet", responsable successivement de sa conception, de son analyse-programmation et de sa mise en oeuvre dans une région-pilote. La généralisation ultérieure de l'application réalisée dans cette région-pilote dépend des résultats obtenus et fait l'objet d'une décision de la Direction Générale. Néanmoins, le chef de projet doit dès le départ considérer que son activité a une vocation nationale donc refuser tout particularisme régional. Il est aidé d'une équipe d'analystes-programmeurs et entouré d'un "groupe de conception" chargé de rédiger le document de "définition des objectifs généraux" puis le "cahier des charges" de l'application, qui sont adressés pour avis à tous les services utilisateurs potentiels et aux chefs de projet des autres applications. Le groupe de conception comprend 5 à 10 personnes représentant les services les plus divers concernés par le projet et comporte obligatoirement un bon analyste attaché à l'application.

II - L'IMPLANTATION GEOGRAPHIQUE D'UN RESEAU INFORMATIQUE PERFORMANT

L'organisation de l'entreprise française des télécommunications repose sur l'existence de 20 régions. Des calculateurs ont été implantés dans le passé au moins dans toutes les plus importantes. On trouve ainsi des machines Bull Gamma 30 à Lyon et Marseille, des GE 425 à Lille, Bordeaux, Toulouse et Montpellier, un GE 437 à Massy, enfin quelques machines Bull 3-90 TT à programmes câblés étaient récemment ou sont encore en service dans les régions de Nancy, Nantes, Limoges, Poitiers et Rouen ; ce parc est essentiellement utilisé pour la comptabilité téléphonique.
A l'avenir, si la plupart des fichiers nécessaires aux applications décrites plus haut peuvent être gérés en temps différé, un certain nombre d'entre eux devront nécessairement être accessibles, voire mis à jour en temps réel : parmi ces derniers le fichier commercial des abonnés, le fichier des renseignements, le fichier des circuits, le fichier technique des abonnés contiendront des quantités considérables d'informations.
Le volume total de caractères à gérer en phase finale par un ordinateur ayant en charge quelques 500 000 abonnés a été estimé à un milliard de caractères au moins. Au moins le tiers des données seront concernées par des traitements en temps réel.
Aucun des calculateurs énumérés plus haut ne permettait d'envisager de tels traitements.
L'intégration progressive de toutes les applications suppose la création d'un support commun pour toutes les informations, une véritable "Banque de données", répartis sur des moyens de traitement nationaux et régionaux, et qui devra rester alimentée, mise à jour en permanence, à partir de la base de l'entreprise, c'est-à-dire les chantiers, les magasins, les guichets des services d'abonnement, les services de personnel etc.
L'étude des différents fichiers à constituer a donc permis de définir les principales caractéristiques du réseau d'ordinateurs nouveaux à mettre en place pour aborder la réalisation du système informatif. L'obligation de faire appel à des ordinateurs de troisième génération, très puissants et dotés de volumineuses mémoires de masse, a conduit à en réduire substantiellement le nombre.
L'implantation de sept centres de calcul interrégionaux constituera un compromis entre : d'une part le désir de réduire le coût économique de l'ensemble, de faciliter la coordination des équipes d'informaticiens ; et d'autre part le refus de créer des centres trop importants difficiles à gérer et à diriger, et posant des problèmes délicats de sécurité. Le regroupement des traitements relatifs à plusieurs régions sur chacun de ces sept centres permettra de leur donner une taille relativement homogène. Chaque centre "gérera" environ un million d'abonnés à la fin du VIème Plan.
La mise en place de ces centres a débuté au début de l'année 1971 : un ordinateur IRIS 80 de la Compagnie Internationale pour l'Informatique a été installé à Toulouse en février ; la même machine vient d'être mise en service au centre de calcul interrégional de Bordeaux.

Photo n° 1 - Document très dense lettre 1,5mm de haut -
Restitution photo n° 9

Fig. 6.8 - Imagem de CCITT4.PBM

Cela est d'autant plus valable que $T\Delta f$ est plus grand. A cet égard la figure 2 représente la vraie courbe donnant $|\phi(f)|$ en fonction de f pour les valeurs numériques indiquées page précédente.

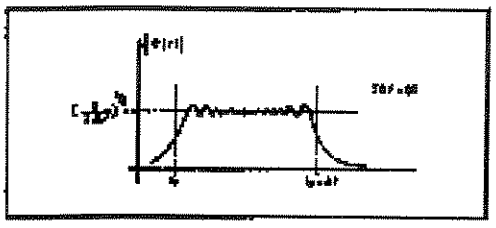


FIG. 2

Dans ce cas, le filtre adapté pourra être constitué, conformément à la figure 3, par la cascade :

— d'un filtre passe-bande de transfert unité pour $f_0 \leqslant f \leqslant f_0+\Delta f$ et de transfert quasi nul pour $f < f_0$ et $f > f_0+\Delta f$, filtre ne modifiant pas la phase des composants le traversant ;

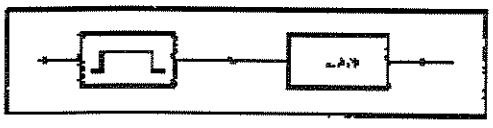


FIG. 3

— filtre suivi d'une ligne à retard (LAR) dispersive ayant un temps de propagation de groupe T_g décroissant linéairement avec la fréquence f suivant l'expression :

$$T_g = T_0+(f_0-f)\frac{T}{\Delta f} \quad (\text{avec } T_0 > T)$$

(voir fig. 4).

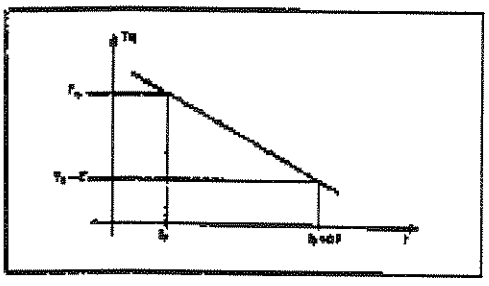


FIG. 4

telle ligne à retard est donnée par :

$$\varphi = -2\pi\int_0^f T_g\,df$$

$$\varphi = -2\pi\left[T_0+\frac{f_0T}{\Delta f}\right]f+\pi\frac{T}{\Delta f}f^2$$

Et cette phase est bien l'opposé de $|\phi(f)|$,

à un déphasage constant près (sans importance) et à un retard T_0 près (inévitable).

Un signal utile $S(t)$ traversant un tel filtre adapté donne à la sortie (à un retard T_0 près et à un déphasage près de la porteuse) un signal dont la transformée de Fourier est réelle, constante entre f_0 et $f_0+\Delta f$, et nulle de part et d'autre de f_0 et de $f_0-\Delta f$, c'est-à-dire un signal de fréquence porteuse $f_0+\Delta f/2$ et dont l'enveloppe a la forme indiquée à la figure 5, où l'on a représenté simultanément le signal $S(t)$ et le signal $S_1(t)$ correspondant obtenu à la sortie du filtre adapté. On comprend le nom de *récepteur à compression d'impulsion* donné à ce genre de filtre adapté : la « largeur » (à 3 dB) du signal comprimé étant égale à $1/\Delta f$, le rapport de compression est de $\dfrac{T}{1/\Delta f} = T\Delta f$

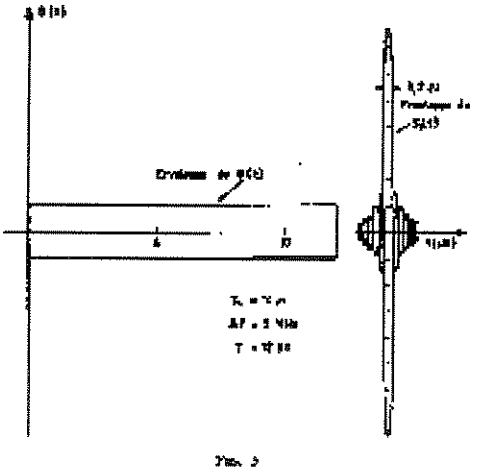


FIG. 5

On saisit physiquement le phénomène de compression en réalisant que lorsque le signal $S(t)$ entre dans la ligne à retard (LAR) la fréquence qui entre la première à l'instant 0 est la fréquence basse f_0, qui met un temps T_0 pour traverser. La fréquence f entre à l'instant $t = (f-f_0)\dfrac{T}{\Delta f}$ et elle met un temps $T_0-(f-f_0)\dfrac{T}{\Delta f}$ pour traverser, et qui la fait ressortir à l'instant T_0 également. Ainsi donc, le signal $S(t)$

Fig. 6.9 - Imagem de CCITT5.PBM

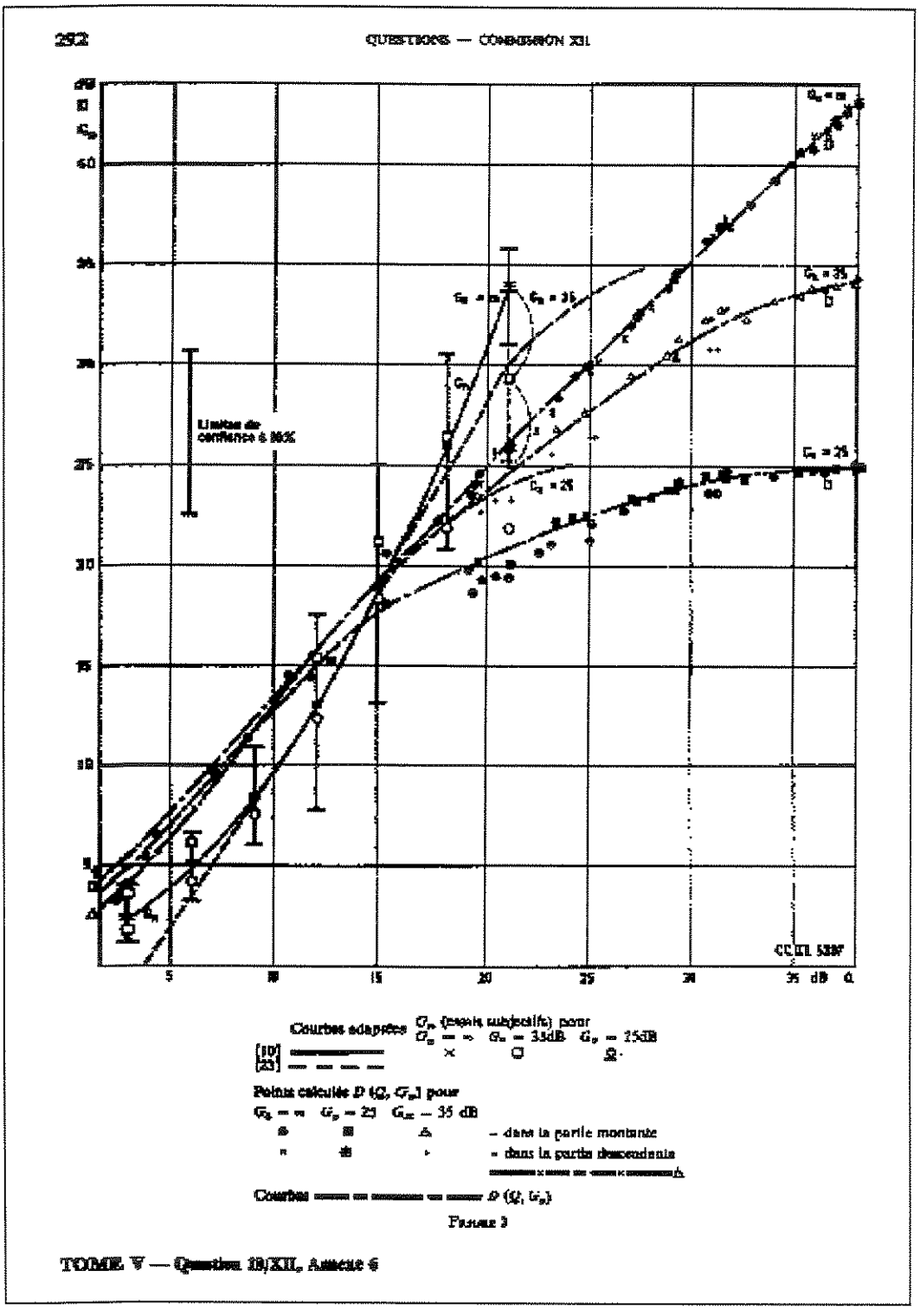


Fig. 6.10 - Imagem de CCITT6.PBM

CCITTの概要

Fig. 6.11 - Imagem de CCITT7.PBM

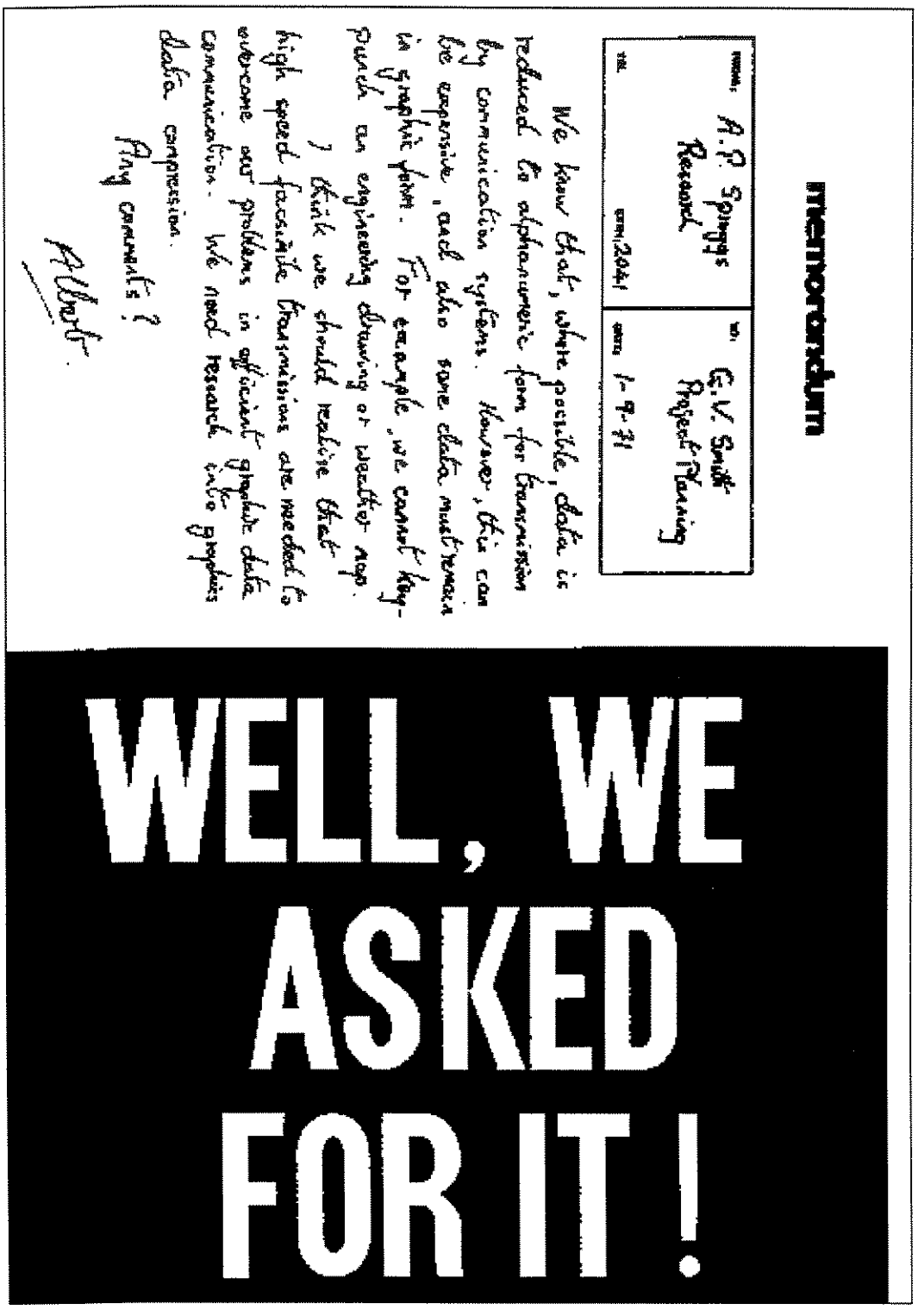


Fig. 6.12 - Imagem de CCITT8.PBM

- **Segundo Tipo**: Imagens escaneadas pelo ***Scanner*** tipo ***ScanMaker IISP*** com o ambiente ***Adobe Photoshop Limited*** versão 2.5. Utilizou-se o *scan* com o modo: ***Line Art*** e uma resolução de 300 dpi. As imagens foram digitalizadas pelo ***scanner*** com tamanho 791 x 1024 *pixels* de 1 *bit* e são mostradas nas figuras (6.13) e (6.14). A figura (6.15) mostra uma imagem com tamanho 869 x 900 *pixels* de 1 *bit*. A imagem com tamanho 992 x 664 *pixels* de 1 *bit* esta mostrada na figura (6.16). Essas imagens contêm escritas à mão com maiores detalhes.

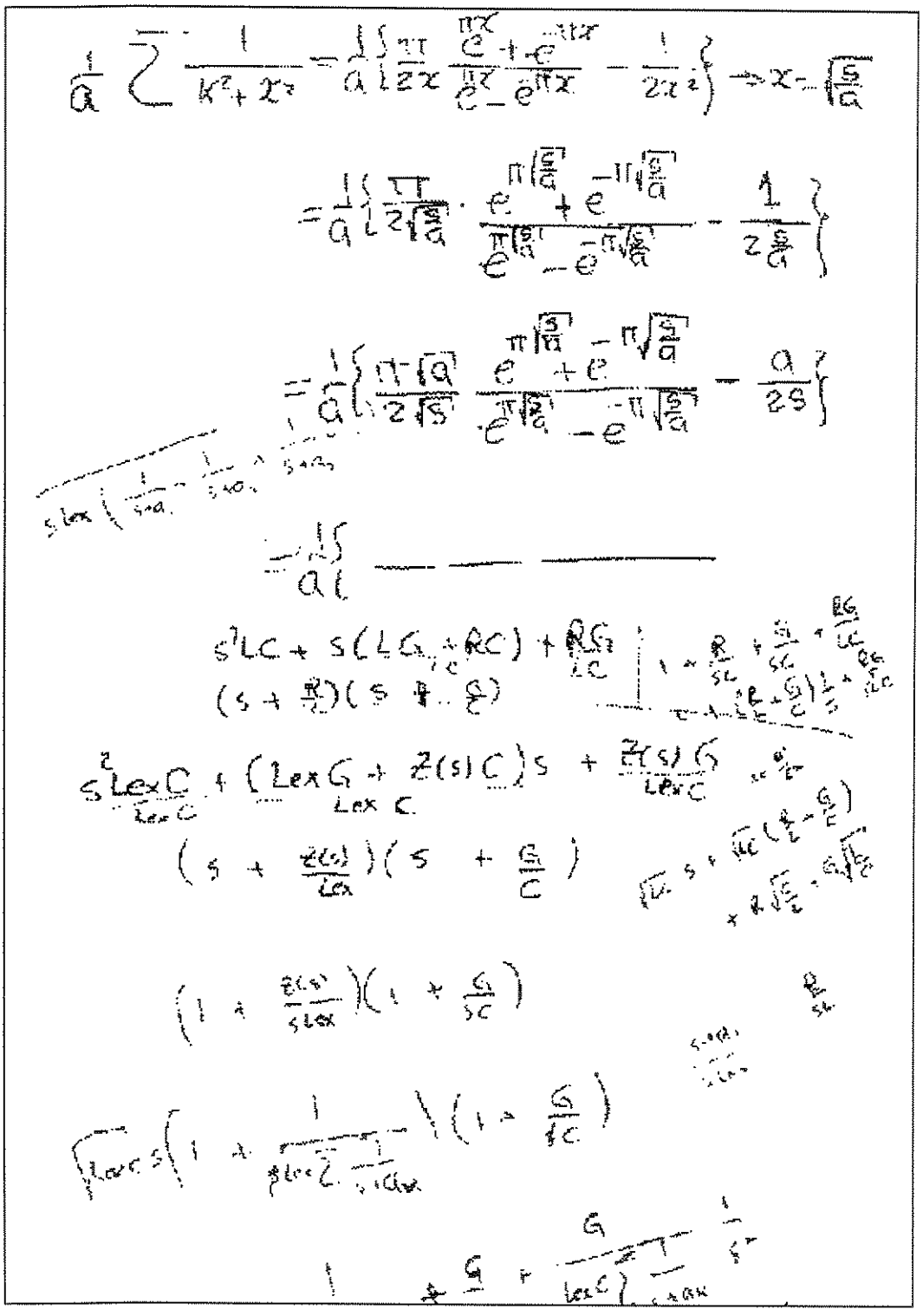


Fig. 6.13 - Imagem de TESTE1.PBM

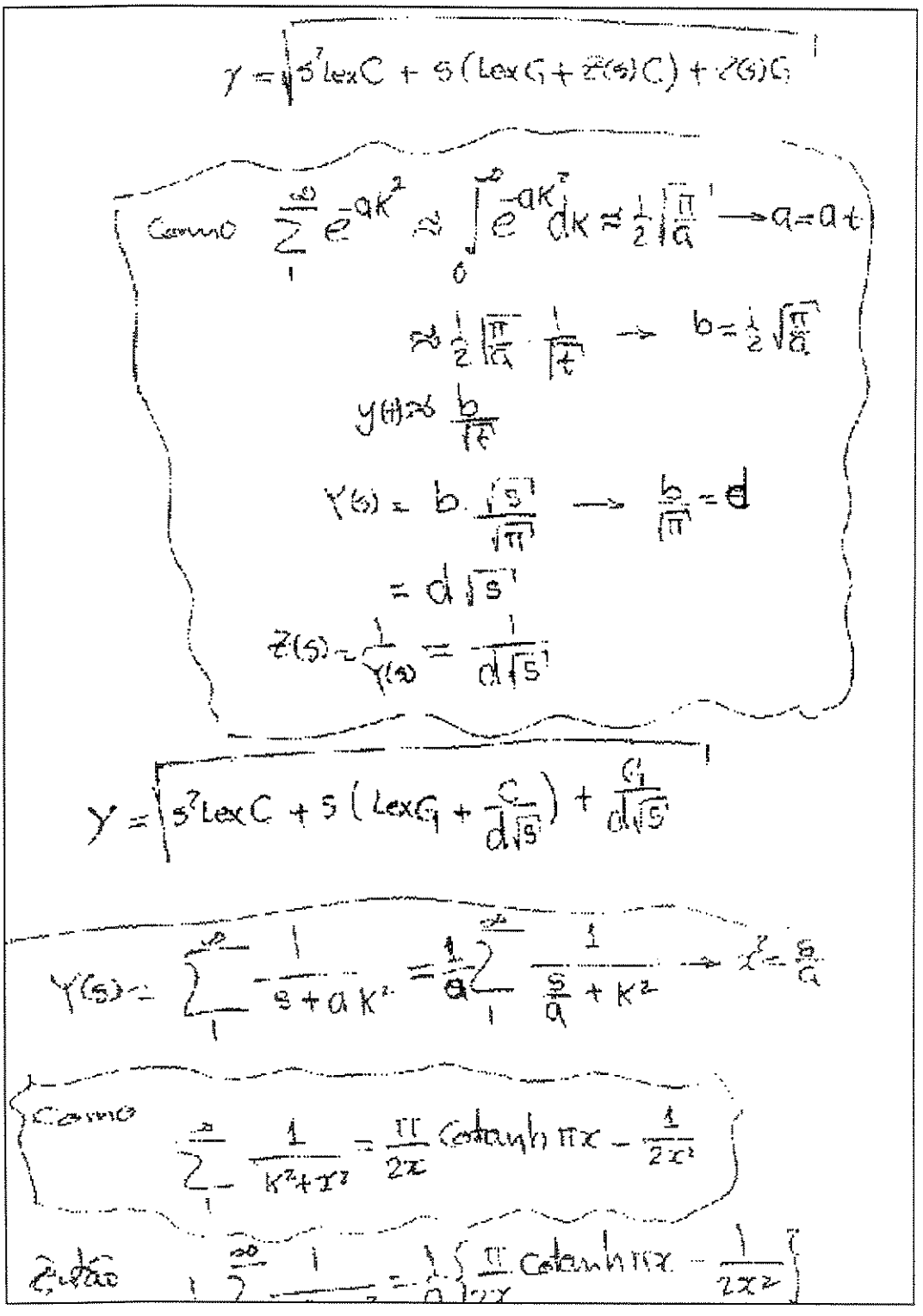


Fig. 6.14 - Imagem de TESTE2.PBM

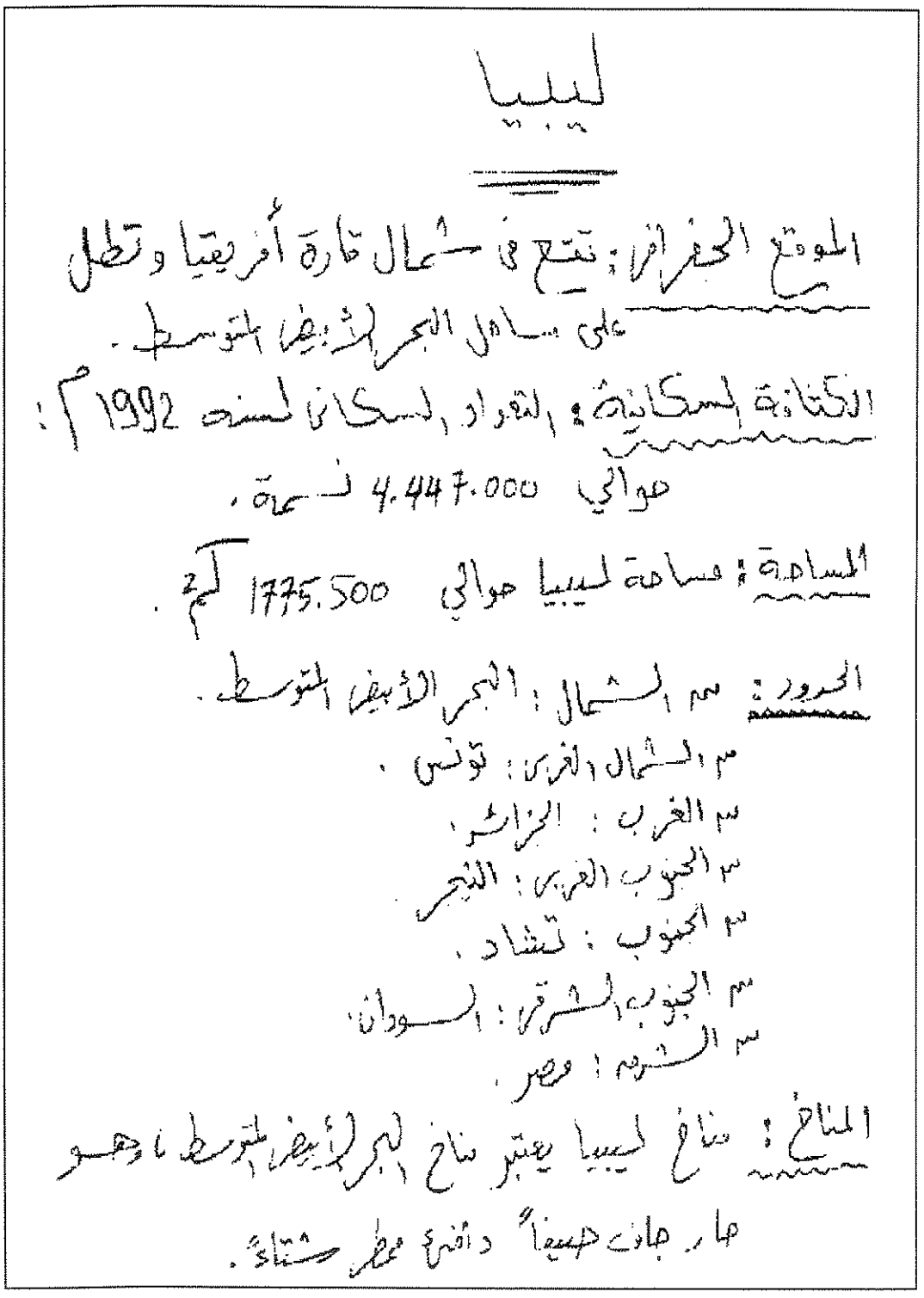


Fig. 6.15 - Imagem de TESTE3.PBM

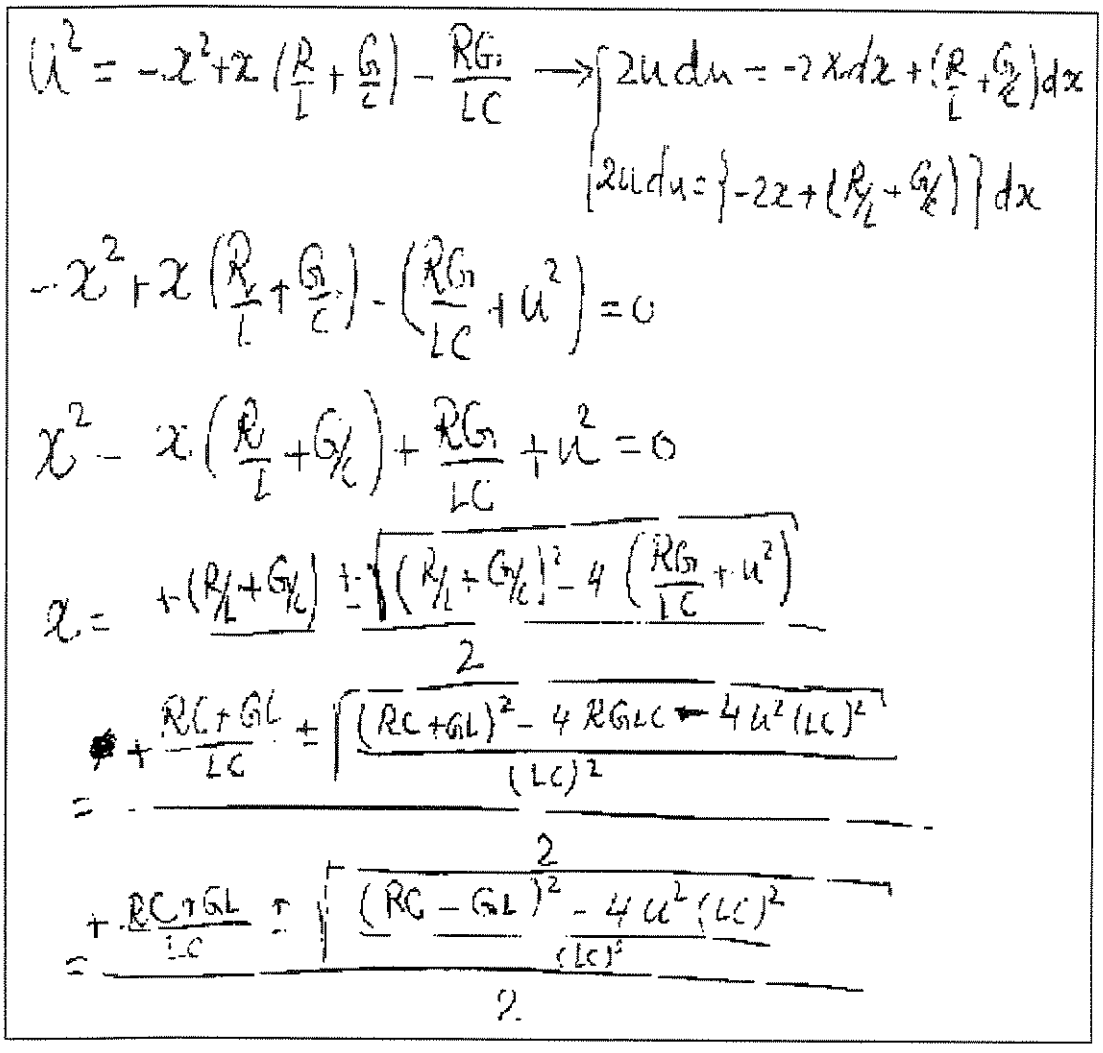
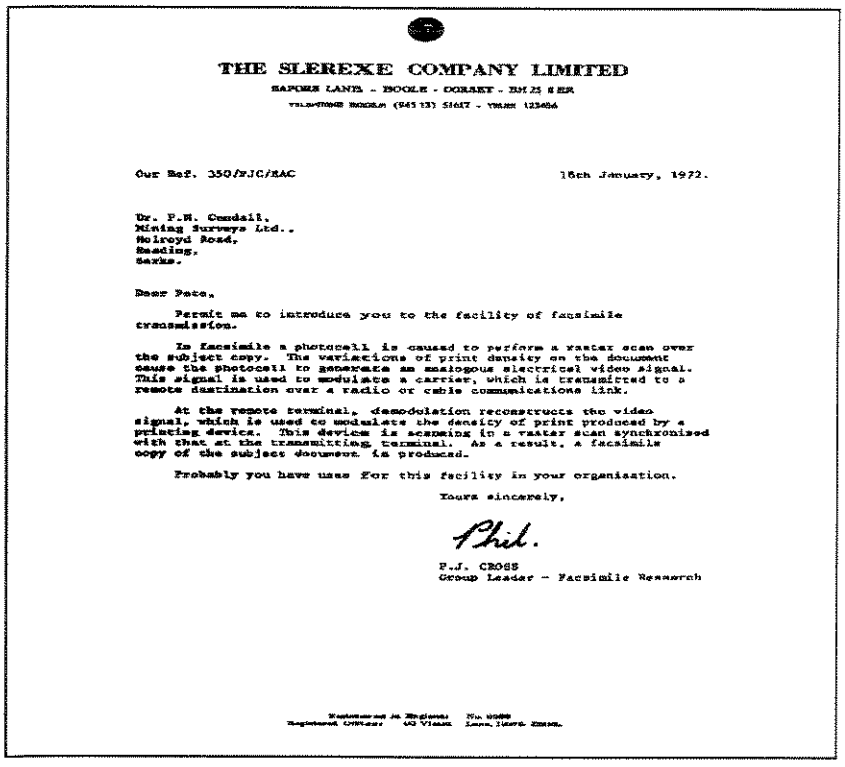


Fig. 6.16 - Imagem de TESTE4.PBM

OBSERVAÇÃO IMPORTANTE : ***As imagens decodificadas são exatamente iguais àquelas que foram codificadas. De fato, a codificação proposta neste trabalho é aquela sem perdas. Dessa forma, as mesmas figuras servem tanto para a imagem processada como para a imagem recuperada. A figura (6.17) é uma amostra de imagem recuperada que é exatamente igual a imagem fonte CCITT1.PBM.***



THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

Our Ref. 350/PJC/EAC 18th January, 1972.

Dr. P.R. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

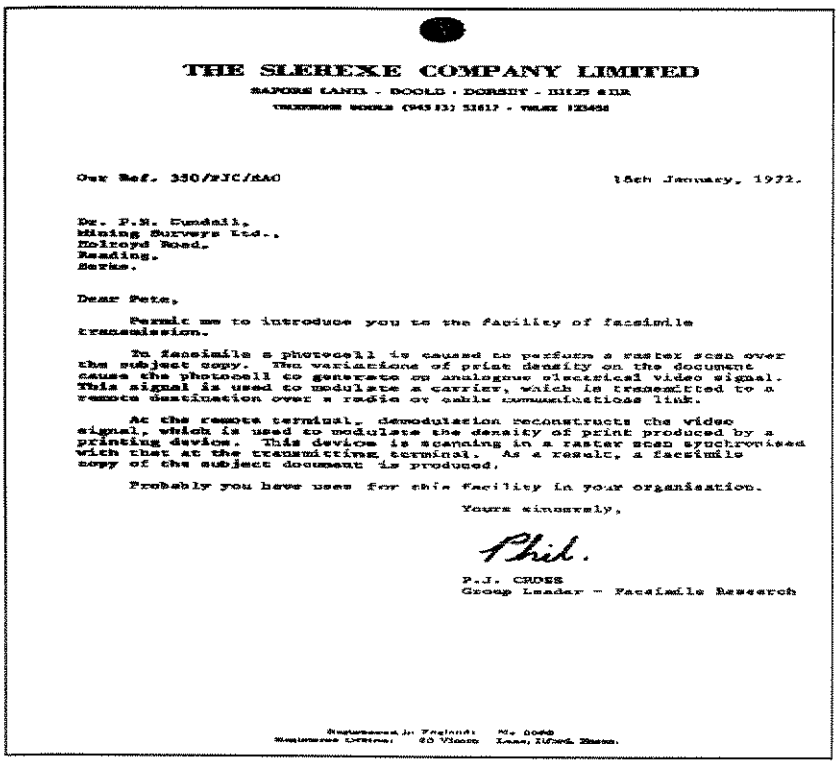
Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Fig. 6.17a - Imagem recuperada de CCITT1.PBM



THE SLEREXE COMPANY LIMITED

SAPORS LANE - BOOLE - DORSET - BH25 8ER

Our Ref. 350/PJC/EAC 18th January, 1972.

Dr. P.R. Cundall,
Mining Surveys Ltd.,
Holroyd Road,
Reading,
Berks.

Dear Pete,

Permit me to introduce you to the facility of facsimile transmission.

In facsimile a photocell is caused to perform a raster scan over the subject copy. The variations of print density on the document cause the photocell to generate an analogous electrical video signal. This signal is used to modulate a carrier, which is transmitted to a remote destination over a radio or cable communications link.

At the remote terminal, demodulation reconstructs the video signal, which is used to modulate the density of print produced by a printing device. This device is scanning in a raster scan synchronised with that at the transmitting terminal. As a result, a facsimile copy of the subject document is produced.

Probably you have uses for this facility in your organisation.

Yours sincerely,

Phil.

P.J. CROSS
Group Leader - Facsimile Research

Fig. 6.17b - Imagem original de CCITT1.PBM

6.3.1 Formato de Imagens de Teste

O algoritmo do *ABAC* recebe somente imagens digitais binárias (imagens de 2 níveis). Caso a imagem não esteja na forma binária, é necessário utilizar um ambiente que faz a conversão para um formato binário.

Neste trabalho, utilizamos o ambiente *xv* versão 3.00 (John Bradley, Mar. 1993) da estação ***SunOS***, para visualizar as imagens digitais e convertê-las para o formato binário do tipo ***PBM*** (“***Portable BitMap***”). Esse formato se chama padrão ***ASCII*** e tem as seguintes características:

- Na primeira linha tem-se o número mágico “***magic number***” para identificar o tipo de formato do arquivo. Esse número é composto por dois caracteres iguais a “P1”;
- Na segunda linha, tem-se um comentário que começa com o símbolo “#”;
- Na terceira linha, tem-se o número da largura dos *pixels* da imagem seguido pelo número da altura;
- As linhas seguintes começam com os *pixels* da imagem. Cada linha tem 35 *pixels* de valores “0”s” e “1’s” com um espaço entre cada dois *pixels*. As linhas não devem ser maiores do que 70 caracteres (incluindo-se os espaços).

A figura (6.18) mostra a forma do formato binário *ASCII* do tipo *PBM*, onde ***cada pixel da imagem representa somente um bit***.

```
P1
# Versão 2, 27-06-1996
1728 2376
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1
1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1
1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1
1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0
                    ⋮                    ⋮                    ⋮
1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1
```

Fig. (6.18) - Estrutura da imagem do formato binário ASCII

Pode-se utilizar também outra forma de formato binário que se chama ***RAW***. Esse formato tem as seguintes diferenças em relação ao formato *ASCII*:

- O número mágico é "P4" ao invés de "P1";
- Os *pixels* da imagem estão em forma de caracteres, ou seja, os *pixels* são concatenados em *bytes*. Cada *byte* tem 8 *bits* (8 *pixels*);
- Não há espaço entre os *bits*.

A vantagem da imagem que utiliza o formato *RAW* é seu tamanho menor do que ao daquele que utiliza o formato *ASCII*. Como conseqüência, o processo de codificação é mais rápido. O formato *RAW* não é utilizado neste trabalho.

Antes do algoritmo de codificação iniciar a leitura dos *pixels* da imagem, os *pixels* são armazenados numa área da memória tal que, os *pixels* de cada linha tenham um número inteiro de *bytes*. Em outras palavras, se a largura da imagem não for múltiplo inteiro de 8 então, alguns *pixels* com valores "0's" devem ser colocados no fim de cada linha. A figura (6.19a) mostra um exemplo de uma imagem que representa a palavra *JBIG* de 35 x 6 *pixels* (sem usar modo progressivo). A figura (6.19b) mostra a representação dos *pixels* na memória em forma binária de tipo *PBM*, onde o *pixel* "0" representa um ponto branco e o *pixel* "1" representa um ponto preto. O modo de leitura dos *pixels* pelo algoritmo de codificação é o modo seqüencial ("***raster mode***") onde os *pixels* são tomados da esquerda para a direita e de cima para baixo.

```
...XXXXXXX...XXXXX....XX....XXXXX..
........XX...XX...X...XX...XX...X..
........XX...XXXXX....XX...XX......
........XX...XX...X...XX...XX.XXX..
...X....XX...XX...X...XX...XX...X..
....XXXXX....XXXXX....XX....XXXX...
```

Fig. 6.19a - Imagem de 35x6 pixels de palavra JBIG

```
00011111 11000111 11000011 00001111 10000000
00000000 11000110 00100011 00011000 10000000
00000000 11000111 11000011 00011000 00000000
00000000 11000110 00100011 00011011 10000000
00010000 11000110 00100011 00011000 10000000
00001111 10000111 11000011 00001111 00000000
```

Fig. 6.19b - A representação da imagem JBIG na memória

6.4 Avaliação do Algoritmo do *ABAC*

Para se verificar a aplicação e a análise do desempenho do algoritmo de compressão *ABAC*, a avaliação é feita utilizando as eqs. (1.1) e (1.4). A tabela (6.1) mostra um resumo dos resultados de tamanhos das imagens de teste originais, dos testes do algoritmo *ABAC* em termos de memória ocupada pela imagem comprimida e a taxa e a porcentagem de compressão obtidas, para cada imagem de teste. Usa-se as equações referidas anteriormente.

Tab. 6.1 - Resultado de avaliação

Imagem de teste	*Tamanho original (bytes)*	*Tamanho comprimido (bytes)*	*Taxa de compressão*	*% compressão*
CCITT1.PBM	8.328.816	14.929	557,90	99,82
CCITT2.PBM	8.328.816	8.820	944,31	99,89
CCITT3.PBM	8.328.816	23.179	359,33	99,72
CCITT4.PBM	8.328.816	55.498	150,07	99,33
CCITT5.PBM	8.328.816	26.542	313,80	99,68
CCITT6.PBM	8.328.816	13.608	612,05	99,84
CCITT7.PBM	8.328.816	57.434	145,02	99,31
CCITT8.PBM	8.328.816	15.254	546,01	99,82
TESTE1.PBM	1.643.163	6.843	240,12	99,58
TESTE2.PBM	1.643.163	6.961	236,05	99,58
TESTE3.PBM	1.586.597	6.449	246,02	99,59
TESTE4.PBM	1.336.247	4.300	310,76	99,68

Capítulo 7

CONCLUSÕES E SUGESTÕES

7.1 Conclusões

O método de codificação aritmética é uma codificação entrópica que utiliza a estatística dos *pixels*. A codificação aritmética é uma das famílias de códigos que compartilha a propriedade de tratar uma cadeia de *bits* como uma magnitude.

Devido à flexibilidade da codificação aritmética, os códigos podem ser obtidos durante o processo de codificação, enquanto os códigos na codificação de *Huffman* são obtidos antes do processo de codificação.

Os códigos de *Huffman* requerem um projeto para calcular palavras código diferentes ("*codeword set*") para os *pixels* com estatísticas diferentes (probabilidades). A codificação aritmética tem capacidade de receber *pixels* também com distribuição de estatísticas diferentes e gerando-se os códigos que afetam diretamente as probabilidades, podendo-se adaptar os códigos com a mudança da estatística. Essa operação é feita durante o processo de codificação. Os *pixels* mais freqüentes ganham códigos mais curtos e os códigos compridos são para os *pixels* menos freqüentes. A codificação de *Huffman* representa a seqüência de *pixels* codificada com palavras código correspondentes aos códigos de cada *pixel*. A codificação aritmética representa a seqüência de *pixels* codificada com uma única palavra código.

A implementação do algoritmo de codificação aritmética binária adaptativa evita a operação de multiplicação através do uso de deslocamento de *bits*. Tal fato faz com que os valores dos registradores *A* e *C* (o tamanho e a base do intervalo) sejam frações binárias. Isso permite o deslocamento para a esquerda dos *bits* contidos nos registradores *A* e *C*.

As eqs. (5.3) e (5.5) apresentam o problema de precisão que provoca a propagação do "*carry-over*" dentro os códigos. O algoritmo da codificação aritmética binária adaptativa tem capacidade de usar uma precisão fixa para controle do tamanho do código, limitando o crescimento dos dígitos nos códigos.

O mecanismo de adaptação de codificação aritmética binária adaptativa dedica-se à determinação do novo estado de estimação de probabilidade adaptativa do *CX* (contexto) atual para codificar o próximo *pixel*.

A estrutura do modelo e a estatística dos *pixels* (probabilidades relativas) compartilham a eficiência da técnica de compressão.

A eficiência do método de compressão de dados usado refere-se à redução da quantidade de dados transmitidos ou armazenados, à redução de custo e à diminuição de tempo de transmissão.

Caso a quantidade de dados seja pequena, então a compressão de dados não é aconselhável. Por outro lado, se a quantidade de dados de um dado serviço é grande, então tem-se um bom candidato para compressão de dados.

A escolha do método de compressão é dependente das características e das aplicações dos dados. A imagem que contem dados de características semelhantes é boa candidata para compressão de dados, em outras palavras, a taxa de compressão ou o fator de compressão é alto.

O estudo mostra que o padrão *JBIG* permite codificar primeiro a imagem de baixa resolução e depois pode-se enriquecê-la com imagens de mais alta resoluções. Também permite-se ao usuário interromper o processo de codificação se desejado.

Os resultados experimentais mostram que o padrão *JBIG* tem vantagens de compressão no seu modo sem perdas para imagens de 2 *tons* em relação às outras técnicas de compressão. De fato, os resultados revelam uma boa taxa de compressão

revelando assim que o padrão *JBIG* é um bom padrão para várias aplicações. Nosso estudo também fornece subsídios para novas propostas.

7.2 Sugestões

- Usar o modo progressivo na técnica de codificação aritmética binária adaptativa, ou seja, dividir a imagem em várias camadas de resolução, codificá-las e enviá-las uma por uma;
- Pode-se usar um algoritmo de codificação progressiva que permita escolher a quantidade de camadas de resolução;
- Usar formato ***RAW*** de tipo ***PBM*** que incrementa a rapidez do algoritmo de codificação;
- Utilizar uma estrutura de modelo de três linhas;
- Adaptar o algoritmo de ***ABAC*** para codificação de imagens com escala de cinza ("***grayscale image***").

FIM

Referências Bibliográficas

[1] CCITT Rec. T.81 | ISO/IEC 10918-1 (JPEG), "**Information technology - Digital compression and coding of continuous - tone still images.**" Requirements and guidelines, 1992.

[2] CCITT Rec. T.82 | ISO/IEC 11544 (JBIG), "**Information technology - Coded representation of picture and audio information - Progressive bi-level image compression.**" 1993.

[3] LANGDON, G. G. Jr. "An introduction to arithmetic coding." **IBM Jornal of Research and Development,** vol. 28, n.2, p. 135-149, March 1984.

[4] LIM, J. S. "**Two - Dimensional signal and image processing.**" Prentice-Hall, 1990.

[5] JAYANT, N. S., NOLL, P. "**Digital coding of waveforms.**" Prentice-Nall, New Jersey, 1984.

[6] JAIN, A. K. "**Fundamentals of digital image processing.**" Prentice-Hall, 1989.

[7] GONZALEZ, R. C., WOODS, R. E. "**Digital image processing.**" Addison-Wesley Publishing Company, September 1993.

[8] HAMMING, R. W. "**Coding and information theory.**" Prentice-Hall, 1986. 2ª edição.

[9] SHANNON, C. E. "The mathematical theory of communication." Partes I e II, **Bell Syst. Tech. J.,** vol. 27, p. 379-423 e 623-656, July 1948.

[10] GALLAGER, R. G. "**Information theory and reliable communication.**" New York: John Wiley and Sons, 1968.

[11] JAIN, A. K. "Image data compression: A review." **Proc. of the IEEE,** vol. 69, n.3, 1981.

[12] HELD, G. "**Compressão de dados técnicas e aplicações considerações de Hardware e Software.**" São Paulo: Érica, 1992.

[13] LYNCH, T. J. "**Data compression techniques and application.**" New York: Van Nostrand Reinhold, 1985.

[14] KALIFA, E. A., YABU-UTI, J. "**Codificação e decodificação de arquivos de fac-símile G3/CCITT.**" Campinas: FEE, UNICAMP, Outubro 1994. Mestrado (Comunicações) - Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, 1994.

[15] LANGDON, G. G. Jr., RISSANEN, J. "Compression of Black-White images with arithmetic coding." **IEEE Trans. Commun,** vol. COM-29, n.6, p. 858-867, June 1981.

[16] SCHALKWIJK, J. P. M. "An algorithm for source coding." **IEEE Trans. Info. Theory,** vol. IT-18, n.3, p. 395-399, May 1972.

[17] RISSANEN, J. J. "Generalized kraft inequality and arithmetic coding." **IBM J. Res. Develop.,** vol. 20, p. 198-203, Mar. 1976.

[18] PASCO, R. "**Source coding algorithms for fast data compression.**" Ph.D. dissertation, Dept. of Elec. Eng., Stanford Univ., Stanford, CA, May 1976.

[19] RISSANEN, J. J., LANGDON, G. G. Jr. "Arithmetic coding." **IBM J. Res. Develop.,** vol. 23, n.2, p. 149-162, Mar. 1979.

[20] JELINEK, F. "Buffer overflow in variable length coding of fixed rate sources." **IEEE Trans. Info. Theory,** vol. IT-14, p. 490-501, May 1968.

[21] ABRAMSON, N. "**Information theory and coding.**" New York: McGraw-Hill, 1963.

Apêndice A: Tabela de Estimação de Probabilidade

ST	LSZ	NMPS	NLPS	SWTCH	ST	LSZ	NMPS	NLPS	SWTCH
0	0x5a1d	1	1	1	56	0x01f8	57	54	0
1	0x2586	2	14	0	57	0x01a4	58	55	0
2	0x1114	3	16	0	58	0x0160	59	56	0
3	0x080b	4	18	0	59	0x0125	60	57	0
4	0x03d8	5	20	0	60	0x00f6	61	58	0
5	0x01da	6	23	0	61	0x00cb	62	59	0
6	0x00e5	7	25	0	62	0x00ab	63	61	0
7	0x006f	8	28	0	63	0x008f	32	61	0
8	0x0036	9	30	0	64	0x5b12	65	65	1
9	0x001a	10	33	0	65	0x4d04	66	80	0
10	0x000d	11	35	0	66	0x412c	67	81	0
11	0x0006	12	9	0	67	0x37d8	68	82	0
12	0x0003	13	10	0	68	0x2fe8	69	83	0
13	0x0001	13	12	0	69	0x293c	70	84	0
14	0x5a7f	15	15	1	70	0x2379	71	86	0
15	0x3f25	16	36	0	71	0x1edf	72	87	0
16	0x2cf2	17	38	0	72	0x1aa9	73	87	0
17	0x207c	18	39	0	73	0x174e	74	72	0
18	0x17b9	19	40	0	74	0x1424	75	72	0
19	0x1182	20	42	0	75	0x119c	76	74	0
20	0x0cef	21	43	0	76	0x0f6b	77	74	0
21	0x09a1	22	45	0	77	0x0d51	78	75	0
22	0x072f	23	46	0	78	0x0bb6	79	77	0
23	0x055c	24	48	0	79	0x0a40	48	77	0
24	0x0406	25	49	0	80	0x5832	81	80	1
25	0x0303	26	51	0	81	0x4d1c	82	88	0
26	0x0240	27	52	0	82	0x438e	83	89	0
27	0x01b1	28	54	0	83	0x3bdd	84	90	0
28	0x0144	29	56	0	84	0x34ee	85	91	0
29	0x00f5	30	57	0	85	0x2eae	86	92	0
30	0x00b7	31	59	0	86	0x299a	87	93	0
31	0x008a	32	60	0	87	0x2516	71	86	0
32	0x0068	33	62	0	88	0x5570	89	88	1
33	0x004e	34	63	0	89	0x4ca9	90	95	0
34	0x003b	35	32	0	90	0x44d9	91	96	0
35	0x002c	9	33	0	91	0x3e22	92	97	0
36	0x5ae1	37	37	1	92	0x3824	93	99	0
37	0x484c	38	64	0	93	0x32b4	94	99	0
38	0x3a0d	39	65	0	94	0x2e17	86	93	0
39	0x2ef1	40	67	0	95	0x56a8	96	95	1
40	0x261f	41	68	0	96	0x4f46	97	101	0
41	0x1f33	42	69	0	97	0x47e5	98	102	0
42	0x19a8	43	70	0	98	0x41cf	99	103	0
43	0x1518	44	72	0	99	0x3c3d	100	104	0
44	0x1177	45	73	0	100	0x375e	93	99	0
45	0x0e74	46	74	0	101	0x5231	102	105	0
46	0x0bfb	47	75	0	102	0x4c0f	103	106	0
47	0x09f8	48	77	0	103	0x4639	104	107	0
48	0x0861	49	78	0	104	0x415e	99	103	0
49	0x0706	50	79	0	105	0x5627	106	105	1
50	0x05cd	51	48	0	106	0x50e7	107	108	0
51	0x04dc	52	50	0	107	0x4b85	103	109	0
52	0x040f	53	50	0	108	0x5597	109	110	0
53	0x0363	54	51	0	109	0x504f	107	111	0
54	0x02d4	55	52	0	110	0x5a10	111	110	1
55	0x025c	56	53	0	111	0x5522	109	112	0
					112	0x59eb	111	112	1

Apêndice B: Programas de Codificação e Decodificação

```
/* -------  Programa de Codificador Aritmetico Binario Adaptativo -------  */


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>

#include "tabelas.h"


                /* ------ Descricao "status" de codificador aritmetico  ----- */

struct code_estado

{
    unsigned char ST[1024];
    unsigned long C;
    long A;
    long SC;
    int CT;
    int BUF;
};


unsigned int lerint(FILE *fpix)

{
    int c;
    unsigned int i;

    while ((c=getc(fpix)) !=EOF && !isdigit(c))
      if (c =='#')
        while ((c=getc(fpix)) !=EOF && !(c == 13 || c == 10));

      if (c != EOF)
      {
        ungetc(c,fpix);
        fscanf(fpix,"%lu",&i);
      }
    return i;
}
```

```
unsigned int tamanho_total = 0;

void BYTEOUT(unsigned int BUF,FILE *PCT)

{
   tamanho_total += 1;
   fprintf(PCT,"%c",BUF);
   return;
}


void INITC(struct code_estado *s)

{
   int i;

   for (i = 0; i < 1024; s->ST[i++] = 0);

   s->C = 0;
   s->A = 0x10000;
   s->SC = 0;
   s->CT = 11;
   s->BUF = -1;              /* buffer vazio */
  return;
}


void MEMORIA(unsigned char *bit, unsigned int alturaM, unsigned int larguraM, unsigned int
largura_tamanho, FILE *fpix)

{
   register unsigned char *p;
   int i,j;

   p = bit;
   for (i = 0; i < alturaM; i++)
      for (j = 0; j < largura_tamanho; j++)
        {
          if ((j < 4) || (j>(larguraM - 3)) || (i == 0) || (i == (alturaM - 1)))
            {
              *p <<= 1;
              *p |= 0;
            }
          else
            {
              *p <<= 1;
              *p |= lerint(fpix) & 1;
            }
          if ((j & 7) == 7)
            ++p;
        }
    return;
}
```

```
void CODIF(struct code_estado *s, unsigned int CX, int PIX, FILE *PCT)

{
    extern short LSZ[];
    extern char NLPS[], NMPS[], SWTCH[];
    register int lsz, ss;
    long TEMP;

    assert(CX >= 0 && CX < 1024);
    ss = s->ST[CX] & 0x7f;
    assert(ss >= 0 && ss < 113);
    lsz = LSZ[ss];

    if ((PIX << 7 ^ s->ST[CX]) & 0x80)
      {
                /* ------ codificar o simbolo de menor ocurrencia LPS ------- */

       if ((s->A -= lsz) >= lsz)
        {
          s->C += s->A;
          s->A = lsz;
        }
       if (SWTCH[ss])
          s->ST[CX] ^= 0x80;

          s->ST[CX] &= 0x80;
          s->ST[CX] |= NLPS[ss];
      }
    else
      {
                /* ------- Codificar o si'mbolo de maiorocurrencia MPS ------ */

       if ((s->A -= lsz) >= 0x8000)
         return;
       if (s->A < lsz)
        {
          s->C += s->A;
          s->A = lsz;
        }
       s->ST[CX] &= 0x80;
       s->ST[CX] |= NMPS[ss];
      }
do
     {
       s->A <<= 1;
       s->C <<= 1;
       --s->CT;
       if (s->CT == 0)
        {
           TEMP = s->C >> 19;
```

```
            if (TEMP > 0xff)
              {
                 if (s->BUF >= 0)
                  {
                    ++s->BUF;
                    BYTEOUT(s->BUF,PCT);
                  }
                 for (; s->SC; --s->SC)
                   BYTEOUT(0x00,PCT);

                 s->BUF = TEMP & 0xff;
                 assert(s->BUF != 0xff);
              }
            else
              if (TEMP == 0xff)
                   ++s->SC;
              else
                {
                   if (s->BUF >= 0)
                     BYTEOUT(s->BUF,PCT);
                   for (; s->SC; --s->SC)
                     BYTEOUT(0xff,PCT);

                   s->BUF = TEMP;
                }
              s->C &= 0x7ffff;
              s->CT = 8;
         }
     } while (s->A < 0x8000);

    return;
}

void COD(struct code_estado *s,unsigned char *bit,unsigned short *espaco_cx,unsigned int
altura,unsigned int largura,unsigned int largura_tamanho, int MOD,FILE *PCT)

{
    unsigned int i,j,cx,PIX;
    unsigned short result,SOM;
    register unsigned short *q_cx;
    register unsigned char *p,*Temp;

    p = bit;
    Temp = bit;
    q_cx = espaco_cx;
    result = 0;
    SOM = 0;
    cx = 0;

    for (j = 0; j < largura_tamanho; j++)
```

```
      if ((j & 7) == 7)
        ++Temp;

    for (i = 0; i < altura; i++)
      {
        result |= *p;   result <<= 8;   ++p;   result |= *p;
        SOM |= *Temp;   SOM <<= 8;   ++Temp;   SOM |= *Temp;
        for (j = 0; j < largura; j++)
          {
            cx |= ((result & 0x7e00) >> 5) + ((SOM & 0xf000) >> 12);
            PIX = ((SOM & 0x800) >> 11) & 1;
            result <<= 1;
            SOM <<= 1;
            CODIF(s,cx,PIX,PCT);
            *q_cx = (unsigned short) cx;
            ++q_cx;
            cx = 0;
            if ((j & 7) == 7)
              {
                result = 0;
                SOM = 0;
                result |= *p;   result <<= 8;   ++p;   result |= *p;
                SOM |= *Temp;   SOM <<= 8;   ++Temp;   SOM |= *Temp;
              }
          }
        result = 0;
        SOM = 0;
        if ((MOD > 0) && (MOD < 6))
          {
            ++p;
            ++Temp;
          }
      }
    return;
}


void FIMC(struct code_estado *s,FILE *PCT)

{
    long TEMP;

    if ((TEMP = (s->A - 1 + s->C) & 0xffff0000) < s->C)
      s->C = TEMP + 0x8000;
    else
      s->C = TEMP;

    s->C <<= s->CT;
    if (s->C > 0x7ffffff)
      {
        if (s->BUF >= 0)
          BYTEOUT(s->BUF + 1,PCT);
```

```
        if (s->C & 0x7fff800)
          for (; s->SC; --s->SC)
            BYTEOUT(0x00,PCT);
      }
    else
      {
        if (s->BUF >= 0)
          BYTEOUT(s->BUF,PCT);

        for (; s->SC; --s->SC)
          BYTEOUT(0xff,PCT);
      }
    if (s->C & 0x7fff800)
      {
        BYTEOUT((s->C >> 19) & 0xff,PCT);
        if (s->C & 0x7f800)
          BYTEOUT((s->C >> 11) & 0xff,PCT);
      }
    return;
}


unsigned char *reserva_char(size_t tamanho)

{
    unsigned char *ptr_p;

    ptr_p = malloc(sizeof(unsigned char) * tamanho);

    if (!ptr_p)
      {
        fprintf(stderr, "Nao tem memoria suficiente para os pixels!\n");
        exit(1);
      }
    return(ptr_p);
}


unsigned short *reserva_short(size_t tamanho)

{
    unsigned short *ptr_cx;

    ptr_cx = malloc(sizeof(unsigned short) * tamanho);

    if (!ptr_cx)
      {
        fprintf(stderr, "Nao tem memoria suficiente para os contextos!\n");
        exit(1);
      }
    return(ptr_cx);
}
```

```
main(argc,argv)

int argc;
char *argv[];

 {

    FILE *fpix, *PCT, *contexto;
    char *fe = argv[1], *fs = argv[2], *fcx = argv[3];
    unsigned int largura,altura,larguraM,alturaM,largura_tamanho,tipo,k;
    unsigned short *espaco_cx;
    unsigned char *bit;
    int MOD,i,j;
    char h;
    struct code_estado s;
    size_t bitmap_tamanho, contexto_tamanho;

    if ((fe != NULL) && (fs != NULL) && (fcx != NULL))
     {
       fpix=fopen(fe,"rb");
       if (!fpix)
        {
          fprintf(stderr,"Nao pode abrir o arquivo de pixels '%s'\n",fe);
          exit(1);
        }
       PCT=fopen(fs,"wb");
       if (!PCT)
        {
          fprintf(stderr,"Nao pode abrir o arquivo de dados compactados '%s'\n",fs);
          exit(1);
        }
       contexto=fopen(fcx,"wb");
       if (!contexto)
        {
          fprintf(stderr,"Nao pode abrir o arquivo de contextos '%s'\n",fcx);
          exit(1);
        }
       if ((k=getc(fpix)) != 'P')
        {
          fprintf(stderr,"O arquivo '%s'nao esta em tipo PBM !\n",fe);
          exit(1);
        }
       tipo=getc(fpix);
       fprintf(PCT,"%c%c",k,tipo);
       while ((k=getc(fpix)) !=EOF && !isdigit(k))
          if (k =='#')
           {
              ungetc(k,fpix);
              while ((k=getc(fpix)) !=EOF && !(k == 13 || k == 10))
                    fprintf(PCT,"%c",k);
```

```
        fprintf(PCT,"\n");
      }
ungetc(k,fpix);
largura = lerint(fpix);
altura = lerint(fpix);
h=getc(fpix);
fprintf(PCT,"%d %d",largura,altura);
printf(" \n\n A imagem com: \n      Largura: %d  e  Altura: %d\n\n",largura,altura);
larguraM = largura + 6;
alturaM = altura + 2;
MOD = larguraM % 8;
if (MOD)
   largura_tamanho = larguraM + (8 - MOD);
else
   largura_tamanho = larguraM;

 bitmap_tamanho = (size_t) ((largura_tamanho / 8) * alturaM);
bit = reserva_char(bitmap_tamanho);

contexto_tamanho = (size_t) (largura * altura);
espaco_cx = reserva_short(contexto_tamanho);

INITC(&s);

switch (tipo)
  {
    case '1':                                          /* formato ASCII de PBM */

              MEMORIA(bit,alturaM,larguraM,largura_tamanho,fpix);
              COD(&s,bit,espaco_cx,altura,largura,largura_tamanho,MOD,PCT);
              break;
    default:
              fprintf(stderr,"A imagem nao esta em formato ASCII !\n");
              exit(1);
  }

FIMC(&s,PCT);
fwrite(espaco_cx,2,contexto_tamanho,contexto);
if (feof(fpix))
 {
   fprintf(stderr, "Fim do arquivo de pixels '%s'nao esperado !\n",fe);
   exit(1);
 }
if (ferror(fpix) || fclose(fpix))
 {
   fprintf(stderr,"Problema durante lendo os pixels de '%s'\n",fe);
   exit(1);
 }
```

```
    if (ferror(PCT) || fclose(PCT))
      {
        fprintf(stderr,"Problema durante escrevendo dados compactados em '%s'\n",fs);
        exit(1);
      }

    if (feof(contexto))
      {
        fprintf(stderr, "Fim do arquivo de contextos '%s'nao esperado !\n",fcx);
        exit(1);
      }
    if (ferror(contexto) || fclose(contexto))
      {
        fprintf(stderr,"Problema durante escrevendo os contextos em '%s'\n",fcx);
        exit(1);
      }
    free(bit);
    free(espaco_cx);
    printf("\n\n      FIM DO PROCESSO DE CODIFICACAO ");
  }
 else
    printf(" -----> 'aconteceu um erro' ! \n\n \
                    Por favor, entre com nomes certos dos arquivos de: \
                     - Entrada de tipo PBM. \
                     - Contexto de extensao CX. \
                     - Saida de extensao PCT. \n\n");
} /* main */
```

```
/* -------  Programa de Decodificador Aritmetico Binario Adaptativo -------  */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>

#include "tabelas.h"


                /* ------ Descric,ao "status" de decodificador aritmetico  ----- */

struct decode_estado

{
   unsigned char ST[1024];
   unsigned long C;
   long A;
   int CT;
};


void LERBYTE(struct decode_estado *s,FILE *PCT,char *fe)

{
   unsigned long k;

   if ((k = getc(PCT)) == EOF)
    {
      k = 0;
      s->C |= (unsigned long) k << 8;
      s->CT = 8;
    }
   else
    {
      s->C |= (unsigned long) k << 8;
      s->CT = 8;
    }
   return;
}


void INITD(struct decode_estado *s,FILE *PCT,char *fe)

{
   int i;

   for (i = 0; i < 1024; s->ST[i++] = 0);
   s->CT = 0;
   s->C = 0;
```

```
    LERBYTE(s,PCT,fe);
    s->C <<= 8;
    LERBYTE(s,PCT,fe);  s->C <<= 8;
    LERBYTE(s,PCT,fe);
    s->A = 0x1O000;
    return;
}


void RENORMDEC(struct decode_estado *s,FILE *PCT,char *fe)

{
    do
     {
        if (s->CT == 0)
        LERBYTE(s,PCT,fe);
        s->C <<= 1;
        s->A <<= 1;
        --s->CT;
     } while (s->A < 0x8000);
    if (s->CT == 0)
      LERBYTE(s,PCT,fe);
    return;
}


int DECODIF(struct decode_estado *s,FILE *PCT,char *fe,unsigned short CX)

{
    extern short LSZ[];
    extern char NMPS[], NLPS[], SWTCH[];
    register int lsz, ss;
    register unsigned char *st;
    int PIX;

    ss = s->ST[CX] & 0x7f;
    assert(ss < 113);
    lsz = LSZ[ss];
    if ((s->C >> 16) < (s->A -= lsz))
       if (s->A >= 0x8000)
        {
           PIX = s->ST[CX] >> 7;
           return PIX;
        }
       else
        {
                              /* ---------- MPSDECODIF ---------- */
           if (s->A < lsz)
            {
               PIX = 1 - (s->ST[CX] >> 7);
```

```
                if (SWTCH[ss])
                    s->ST[CX] ^= 0x80;
                s->ST[CX] &= 0x80;
                s->ST[CX] |= NLPS[ss];
            }
          else
            {
                PIX = s->ST[CX] >> 7;
                s->ST[CX] &= 0x80;
                s->ST[CX] |= NMPS[ss];
            }
        }
    else
      {
                            /* ---------- LPSDECODIF ---------- */
        if (s->A < lsz)
          {
            s->C -= s->A << 16;
            s->A = lsz;
            PIX = s->ST[CX] >> 7;
            s->ST[CX] &= 0x80;
            s->ST[CX] |= NMPS[ss];
          }
        else
          {
            s->C -= s->A << 16;
            s->A = lsz;
            PIX = 1 - (s->ST[CX] >> 7);
            if (SWTCH[ss])
                s->ST[CX] ^= 0x80;
            s->ST[CX] &= 0x80;
            s->ST[CX] |= NLPS[ss];
          }
      }
    RENORMDEC(s,PCT,fe);
    return PIX;
}


void DCOD(struct decode_estado *s,unsigned short *espaco_cx,unsigned int altura, unsigned
int largura,FILE *PCT,FILE *pix_final,char *fe)

{
    register unsigned short cx,*q_cx;
    int c_linha,i,j,PIX;

    c_linha = 0;
    q_cx = espaco_cx;
    for (i = 0; i < altura; i++)
```

```
      for (j = 0; j < largura; j++)
        {
          cx = *q_cx;
          ++q_cx;
          PIX = DECODIF(s,PCT,fe,cx);
          if (c_linha < 34)
            {
              fprintf(pix_final,"%d ",PIX);
              ++c_linha;
            }
          else
            {
              fprintf(pix_final,"%d\n",PIX);
              c_linha = 0;
            }
        }
    return;
}


unsigned short *reserva_short(size_t tamanho)

{
    unsigned short *ptr_cx;

    ptr_cx = malloc(sizeof(unsigned short) * tamanho);
    if (!ptr_cx)
      {
        fprintf(stderr, "Na~o tem memoria suficiente para os contextos!\n");
        exit(1);
      }
    return(ptr_cx);
}


main(argc,argv)

 int argc;
 char *argv[];

  {
     FILE *PCT, *pix_final, *contexto;
     char *fe = argv[1], *fs = argv[2], *fcx = argv[3];
     unsigned int largura,altura,tipo,k;
     unsigned short *espaco_cx;
     char h;
     struct decode_estado s;
     size_t contexto_tamanho;
```

```
if ((fe != NULL) && (fs != NULL) && (fcx != NULL))
 {
    PCT=fopen(fe,"rb");
    if (!PCT)
     {
          fprintf(stderr,"Nao pode abrir o arquivo de dados compactados '%s'\n",fe);
          exit(1);
     }
    pix_final=fopen(fs,"wb");
    if (!pix_final)
     {
          fprintf(stderr,"Nao pode abrir o arquivo de pixels recuperados '%s'\n",fs);
          exit(1);
     }
    contexto=fopen(fcx,"rb");
    if (!contexto)
     {
          fprintf(stderr,"Nao pode abrir o arquivo de contextos '%s'\n",fcx);
          exit(1);
     }
    if ((k=getc(PCT)) != 'P')
     {
          fprintf(stderr,"O arquivo '%s'na~o e' tipo PBM !\n",fe);
          exit(1);
     }
    tipo=getc(PCT);
    fprintf(pix_final,"%c%c\n",k,tipo);
    printf("%c%c\n",k,tipo);
    while ((k=getc(PCT)) !=EOF && !isdigit(k))
      if (k =='#')
        {
          ungetc(k,PCT);
          while ((k=getc(PCT)) !=EOF && !(k==13 || k == 10))
              fprintf(pix_final,"%c",k);
              fprintf(pix_final,"\n");
        }
    ungetc(k,PCT);
    fscanf(PCT,"%d",&largura);
    fscanf(PCT,"%d",&altura);
    fprintf(pix_final,"%d %d\n",largura,altura);
    printf("\n\n A imagem recuperada tem: \n    Largura =  %d  e  Altura = %d\n\n", \
                                          largura,altura);
    contexto_tamanho = (size_t) (largura * altura);
    espaco_cx = reserva_short(contexto_tamanho);

    fread(espaco_cx,2,contexto_tamanho,contexto);

    INITD(&s,PCT,fe);
```

```
        switch (tipo)
          {
            case '1':                                   /* formato ASCII tipo PBM */

                    DCOD(&s,espaco_cx,altura,largura,PCT,pix_final,fe);
                    break;
            default:
                    fprintf(stderr,"Tipo P%c (PBM) insuportado!\n",tipo);
                    exit(1);
          }
        if (ferror(PCT) || fclose(PCT))
          {
              fprintf(stderr,"Problema durante lendo os dados compactados de '%s'\n",fe);
              exit(1);
          }
        if (ferror(pix_final) || fclose(pix_final))
          {
              fprintf(stderr,"Problema durante escrevendo os pixels recuperados \
                                                em '%s'\n",fs);
              exit(1);
          }
        if (feof(contexto))
          {
              fprintf(stderr, "Fim do arquivo de contextos '%s'nao esperado !\n",fcx);
              exit(1);
          }
        if (ferror(contexto) || fclose(contexto))
          {
              fprintf(stderr,"Problema durante lendo os contextos de '%s'\n",fcx);
              exit(1);
          }
        free(espaco_cx);
        printf("\n\n      FIM DO PROCESSO  DE  DECODIFICACAO ");
      }
    else
      printf(" -----> 'aconteceu um erro' ! \n\n \
                      Por favor, entre com nomes certos dos arquivos de: \
                       - Entrada de extensao PCT. \
                       - Contexto de extensao CX. \
                       - Saida de extensao PBM. \n\n");
} /* main */
```

Trabalhos Realizados

- Iano, Y., Khalifa, E. A., Abushaala, A. M. "Digital Television: HDTV/NTSC/PAL-M - A review." In: SIMPÓSIO NIPO-BRASILEIRO DE CIÊNCIA E TECNOLOGIA, 100, 1995, São Paulo. **Informatics and Telecommunications.** São Paulo: Academia de Ciências do Estudo de São Paulo Sociedade Brasileira de Pesquisadores Nikkeis. Aug. 1995. p. 185-202.

- Khalifa, E. A., Abushaala, A. M., Iano, Y. "Codificação de arquivos fac-símile digital do grupo 3 (G3)/CCITT." In: ELECTRO'95: CONGRESSO CHILENO DE INGENIERIA ELECTRICA, XI, 1995, Punta Arenas-CHILE. **Automatizacon de Processos y de Oficinas.** Punta Arenas-CHILE: Universidad de Magallanes, Nov. 1995. vol. II, p. C-006 - C-010.

- Abushaala, A. M., Khalifa, E. A., Iano, Y. "Compatibility analysis between conventional TV and HDTV." In: ELECTRO'95: CONGRESSO CHILENO DE INGENIERIA ELECTRICA, XI, 1995, Punta Arenas-CHILE. **Sistemas de Comunicaciones.** Punta Arenas-CHILE: Universidad de Magallanes, Nov. 1995. vol. II, p. I-030 - I-034.

- Abushaala, A. M., Iano, Y. "Considerations on JBIG standrad: Is it a good choice for Brazilian transmission systems?." In: INTERCON'96: INTERNATIONAL ELECTRONIC ENGINEERING CONGRESS, 3, 1996, Trujillo-PERU: Antenor Orrego Private University, Aug. 1996.

- Iano, Y., Almeida, A. M., Fançony, A. M., Paschoarelli, A. C., Pelaes, E. G., Khalifa, E. A., Abushaala, A. M., Maria, Y. B., Batista, R. Q. "Avaliação de sistemas - Linha de Pesquisa em sistemas audiovisuais avançados." RELATÓRIO TÉCNICO, contrato Telebrás 387/96, Processamento Digital PT03, RT-244. Pub. em: FEEC 13/96, Abril 1996, Campinas-BRASIL.

- Iano, Y., Almeida, A. M., Fançony, A. M., Paschoarelli, A. C., Pelaes, E. G., Khalifa, E. A., Abushaala, A. M., Maria, Y. B., Batista, R. Q. "Programas para avaliação de sistema - Software básico e Específico - Linha de Pesquisa em sistemas audiovisuais avançados." RELATÓRIO TÉCNICO, contrato Telebrás 387/96, Processamento Digital PT03, RT-244. Pub. em: FEEC 13/96, Abril 1996, Campinas-BRASIL.