"RECUPERAÇÃO DE INFORMAÇÃO PARA MICROPROCESSADORES"

Eduardo Dória Silva

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

Prof. Guilherme Chagas Rodrigues

Prof. Nelson Maculan Filho

Prof. Ivan da Costa Marques

Prof. Jayme Luiz Szwarcfiter

RIO DE JANEIRO

ESTADO DO RIO DE JANEIRO - BRASIL

JULHO DE 1976

Aos meus Pais

Luiz Acioli da Silva

Júlia Dória Silva e

Irmãos

AGRADECIMENTOS

Cumpre-me agradecer ao Professor Guilherme Chagas Rodrigues pelo acompanhamento e valiosa orientação da tese, aos colegas Professor Zacharias Ernane das Candeias, Milton Albuquerque por significativas colaborações e a todos que de modo direto ou indiretamente contribuiram para a conclusão deste trabalho.

Às entidades que patrocinaram com o apoio financeiro.

RESUMO

Trata-se do desenvolvimento e implantação de um sistema em "software" para manipulação de arquivos com microprocessadores.

Este trabalho faz parte de um projeto, coordenado pelo Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro para dar suporte ao usuário do terminal inteligente construído com a unidade central de processamento 8008 (INTEL), do qual se deu maior atenção.

Os algoritmos são apresentados para serem aplicados a quaisquer tipos de microprocessadores.

ABSTRACT

This work is concerned with the development and implantation of a software system which works with the files of a microprocessors.

The project, of which this work is one part, was co-ordinated by the Nucleo de Computação Eletrônica of Universidade Federal do Rio de Janeiro, which supported and greatly encouraged the use of the inteligent terminal built with the Central Processing Unit 8008 (INTEL).

The algorithms are presented to be applied to any microprocessors type.

INDICE

I - INTRODUÇÃO

1.1-Utilidade

O sistema de "hardware" do terminal inteligente foi desenvolvido para ser capaz de efetuar processamento local podendo eventualmente ser ligado a um computador central.

Aqui está desenvolvido um sistema em "software" necessário ao terminal inteligente quando usado como um pequeno computador pois, todo o processamento é efetuado através de seu microprocessador INTEL - 8008 [20]. Portanto este sistema capacita o de "hardware" para aplicações necessárias na indústria e comércio de pequeno e médio porte, tais como : emissão de relatórios, faturamento, controle de estoque, contabilidade, etc.

Basicamente, opera-se consultando um arquivo de entrada e gerando outro de saída, submetendo o primeiro a restrições impostas pelo interesse do usuário, através de expressões do cálculo de relação [6], apresentadas sob forma de tabela que está descrita em III-5 com o título de "Tabela de restrições".

Em face de o terminal inteligente ser carente de um compilador que proporcionasse ao uso de uma linguagem mais cômoda, o trabalho prático está apresentado no "assembler" simbólico do próprio INTEL. Para isto fez-se uso do Sistema Operacional de Simulação [25] implantado no computador B-6700 para tarefas de depuração e para se obter módulo objeto em linguagem de máquina.

O capítulo dedicado a sugestões motiva uma flexibilidade maior, pela ampliação do uso de operações com arquivos.

Na descrição dos algoritmos usou-se uma linguagem parecida com o algol por assim achar mais potente, cômoda e inteligível.

1.2 - Diretrizes do projeto

"A priori", as diretrizes de todo trabalho são as normas de procedimantos principais que nortearam no desenvolvimento do projeto. As propriedades mais importantes do sistema são:

a) ser de fácil manipulação do usuário - focalizada como a melhor das ideias apresentadas no capítulo II;

b) ser serialmente reusável permitindo, assim, ser chamado em pontos diferentes de um mesmo programa feito pelo usuário;

c) ser modular, onde com apenas a modificação de um módulo, tem-se um novo sistema que decodifica registros permitindo como aplicação, que o usuário faça perguntas quantitativas - no capítulo IV é dado um exemplo.

d) ser qualitativo e de relação - descrito no quinto capítulo.

1.3 - Linguagem utilizada na Redação e nos Algoritmos

Os termos técnicos exibidos no texto estão de acordo com a linguagem utilizada em análise de sistemas [13,5].

A respeito dos algoritmos, serão apresentados, via de regra, através dos já conhecidos comandos da linguagem algol [3]. Tendo em vista a mesma ser pouco descritiva, usa-se como complemento desta, termos da linguagem oficial do Brasil de acordo com a lei 5765 de 18 de Dezembro de 1971 [11] a fim de tornar os algoritmos concisos e mais bem claros [21]. Por conveniência do autor serão dadas explicações técnicas adicionais , quando necessário for, após cada algoritmo.

Alguns comandos serão introduzidos e modificados do algol segundo Knuth [23] para facilitar a apresentação dos algoritmos de forma estruturada [9,29]. Isto minora o tempo de execução, facilita a decumentação e depuração dos programas. [23].

O símbolo "FI" é usado "ad hoc" para fechar bloco para todos os comandos "IF" tornando "BEGIN" e "END desnecessários entre "THEN" e "ELSE".

Um comando indicador de consequência é apresentado com a seguinte metalinguagem

```
LOOP UNTIL <consequência> 1 OR ... OR
           <consequência> N :
           <lista de comando> 0;
```

REPEAT;

THEN <consequência> 1 => <lista de comando> 1

<consequência> N => <lista de comando> N

FI

O novo comando <consequência> significa que ocorreu a consequência : tal comando é permitido sòmente dentro de uma <lista de comando> 0 de uma estrutura UNTIL a qual declara aquela consequência.

II - ASPECTO GERAL DO PROBLEMA

2.1 - Método de trabalho

Para solucionar o problema proposto, considerou-se quatro opções:

- Construir um compilador em linguagem "assembler" da 8008;
- Implantar um sistema conhecido;
- Construir um compilador em linguagem de alto nível;
- Construir um sistema que trata uma tabela em linguagem da 8008.

2.1.1- Construir um compilador em linguagem "assembler" da 8008.

Esta ideia nasceu com o tratamento de expressões de relação onde seriam submetidas a um compilador, via do qual seria construida uma tabela para posterior processamento. Esta definição seria muito cômoda para o usuário pois, os programas seriam escritos de modo bastante claro através de uma linguagem praticamente corrente. O alcance de maior número de aplicações com conjunto de dados é o motivo mais técnico desta ideia. Entretanto, ela foi abandonada em virtude de sua grande carga de trabalhos. A razão mais forte que motivou a desistência, contudo, foi o limite de memória do terminal inteligente que pode dispor no máximo de 16 k-bytes de 8 bits/byte de memória. Para

o caso de um compilador residente, em virtude da carência de uma unidade periférica de disco, a memória é insuficiente.

2.2 - Implantar um sistema conhecido

A implantação do " IMS " [18] " BMS " [4] ou de um sistema mais novo e eficiente "SQUARE" [2] diminuiria um pouco as dificuldades anteriores pois não careceria da definição de linguagens mas os problemas de memória e periféricos são pertinentes.

O leitor poderá estar imaginando a possibilidade de se anexar um periférico de disco ao terminal, o que não constitue uma tarefa difícil. Aqui, tem-se como meta, atingir aplicações para terminais padrões, isto é, montado com disponibilidade de canal, chaves, luzes, vídeo, teclado, unidade de fita "cassette" e com memória no seu limite mínimo de 4 K-bytes por restrição de "hardware".

2.3 - Construir um compilador em linguagem de alto nível

Pensou-se também na construção de um compilador desenvolvido em um computador de porte maior. Assim sendo poder-se-ia realizar o sistema em uma linguagem de alto nível onde o compilador trataria um programa feito pelo usuário na nova linguagem e forneceria em cartões perfurados em formato de linguagem de máquina do processador 8008 o programa objeto. Naturalmente que seria necessário o carregamento deste programa objeto na memória ou qualquer outra unidade que possa guardar programas no terminal. Esta ideia foi uma das piores. Pode-se observar que o usuário para empregar o terminal como um pequeno computador autossuficiente dependeria obrigatoriamente de um sistema de maior porte. Isto não deixa dúvidas.

2.4 - Construir um sistema que trata uma tabela em linguagem "assembler" da 8008.

Neste caso o trabalho é desenvolvido levando-se em consideração que o usuário construirá um programa em linguagem "assembler" simbólica da 8008, com as seguintes características:

- definição de uma área de trabalho endereçada através de um descritor;
- definição de uma área para leitura do registro pertencente ao arquivo de origem com seu respectivo descritor de cadeia;
- uma tabela de restrições através da qual o usuário informa a maneira de se tratar o arquivo, e
- um parâmetro indicador do endereço do arquivo desejado.

Para todos os casos, os sistemas seriam desenvolvidos para arquivos armazenados em fita "cassette"; entretanto, a filosofia não seria mudada se o mesmo estiver guardado em outra unidade periférica desde que se tenha a respectiva rotina de buscas.

2.5 - Opção escolhida

Dentre as principais ideias apresentadas não há dúvidas quanto à última, isto é, a construção de um sistema que trata uma tabela em linguagem "assembler" da 8008 porque satisfaz melhor às condições de comodidade de implantação, conforto para o usuário, além de preencher o requisito mais importante, ou seja, a restrição de memória.

Outro argumento a favor desta opção é que a mesma pode ser usada como subrotina, dando portanto muito mais flexibilidade ao uso do sistema.

2.6 - Conceitos

Os processos de tratamento de arquivos podem ser encarados como conjuntos de dados. Isto possibilita o seu tratamento através da teoria dos Conjuntos ou, de modo mais preciso, por meio dos conceitos de Relação [7] fazendo-se uso do Cálculo de Relação [6].

Para o que se propõe chegar é feito apenas o uso da relação de restrição, onde o subconjunto obtido pertence "in totum" ao conjunto original.

As restrições estabelecidas têm como finalidade selecionar conjuntos ou particularmente dados.

Formalmente isto pode ser assim definido : chamando-se U o conjunto universo, B será o conjunto selecionado de A se

$$A \in U \quad \wedge \quad B \in U$$

$$\forall\, bi \in B \;\exists \text{ pelo menos um } ai \in A \mid bi \equiv ai$$

$$\text{logo } B \subseteq A$$

Esta expressão a que se chegou para seleção de dados pode ser vista sob vários aspectos dentre os quais aqui está desenvolvido um sistema em "software". Para isto fez-se uso, naturalmente, dos subsistemas já desenvolvidos para o terminal, como é o caso das rotinas de entradas e saídas.

III . DEFINIÇÃO

3.1 - Apresentação

O trabalho está constituido dos seguintes módulos conforme figura 1 :

- rotina de gerência ("FILE");
- leitura e gravação em fita "cassette";
- pré-processamento da tabela de restrições;
- rotina de tratamento de erros e advertências;
- módulos de consersões numéricas, booleanas e de caracteres;
- rotinas interface de preparação para operar variáveis numéricas, booleanas e de caracteres, e
- rotinas de operação, respectivas.

Este projeto faz o sistema modular pela independência de suas rotinas. Elas são estanques ou fazem interface entre rotinas que executam tarefas específicas.

A descrição detalhada e os algorítmos de cada um dos módulos serão apresentados no próximo capítulo e no apêndice, respectivamente.

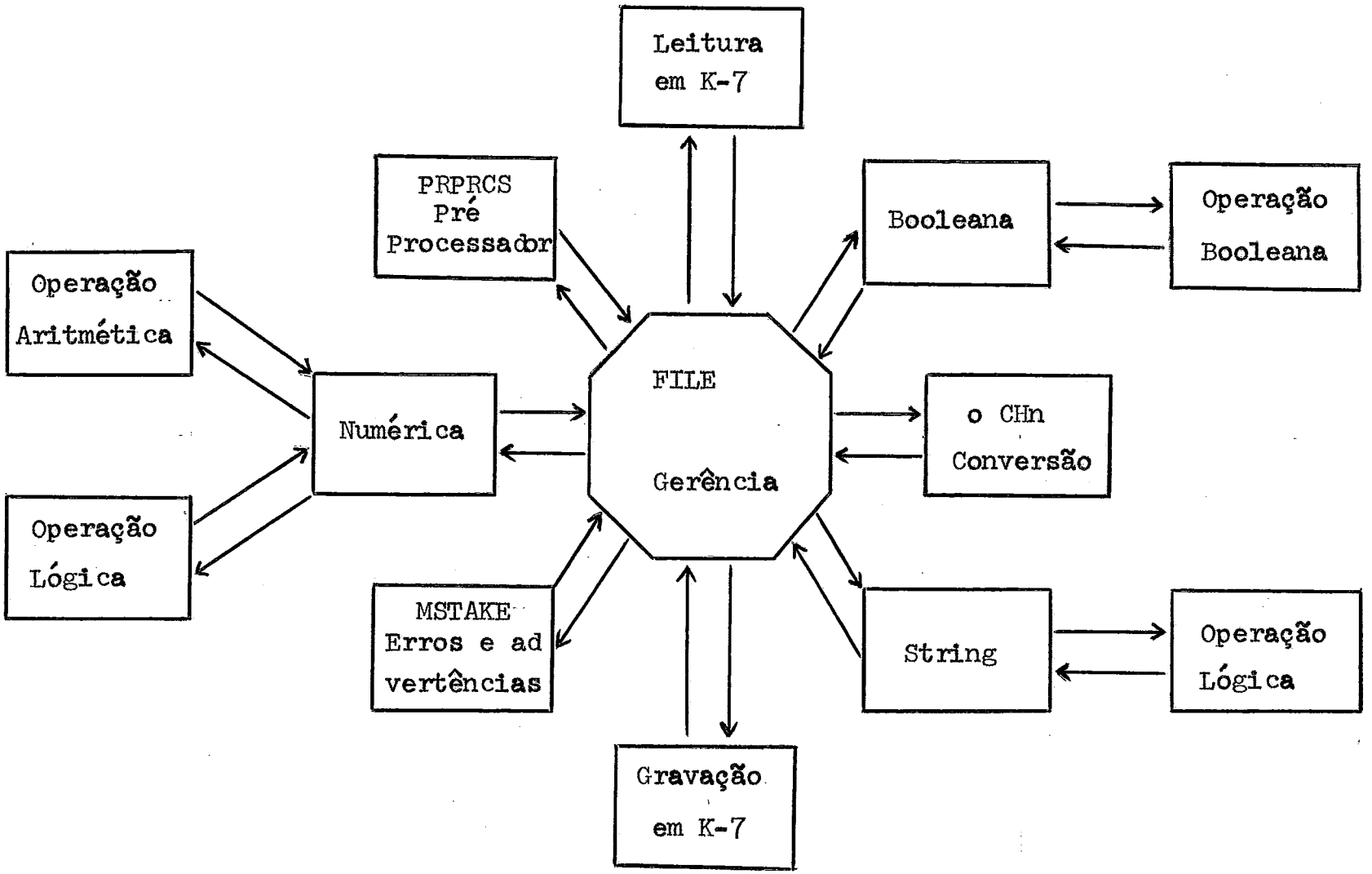

Leitura em K-7
PRPRCS Pré Processador
Booleana
Operação Booleana
Operação Aritmética
Numérica
FILE
Gerência
o CHn Conversão
Operação Lógica
MSTAKE Erros e ad vertências
String
Operação Lógica
Gravação em K-7

FIG. 1

3.2 - Modularidade

Já se falou que o sistema é todo modular para facilitar o usuário adaptá-lo a aplicações específicas. Com isto pode-se retirar ou colocar rotinas de acordo com o trabalho a ser executado, evitando-se o desperdício de memória-condição crítica no terminal. As modificações mais solicitadas dizem respeito às rotinas de conversão.

O sistema opera sob uma base de padrões pré-estabelecidos para executar o programa do usuário, descritos no capítulo IV. Todos os dados fornecidos ao sistema devem ser convertidos inicialmente para posterior processamento. Daí as razões admitidas por ser, no módulo de conversão, onde há maior necessidade de modificações. Eventualmente haverá necessidade até de se definir e programar uma nova rotina.

3.3 - Área de leitura e área de trabalho

Ambas representam memória reservada pelo programa do usuário. A primeira serve para armazenar um registro ou bloco lido directamente da fita de origem, enquanto a área de trabalho guarda vetores numéricos, booleanos ou de caracteres.

3.4 - Descritores

Os descritores aqui referidos têm características e finalidades semalhantes. Transmitem à rotina principal "FILE", endereços e comprimento das áreas de leitura e trabalho : maiores informações podem ser encontradas no manual de referência do usuário do S.O.S. [25] na seção de descritores de cadeia.

3.5 - Tabela de restrições

Vale a pena lembrar que se chama tabela de restrições ao conjunto de informações enviadas ao sistema aqui desenvolvido, construida em um programa do seu usuário.

Algumas formas conhecidas de se escrever expressões são : infixa, posfixa, polonesa, etc. Optou-se pela infixa por ser empregada mais fàcilmente e ser o modo corrente de uso da notação matemática, fato que benificia o usuário.

A desvantagem dessa representação é levar mais tempo de processamento, porém o sistema será limitado por entrada e saída.

Decidiu-se por entrada de tamanho fixo na tabela para simplificar o processamento.

3.6 - Representação dos parâmetros na tabela de restrições

endereço	comprimento	conversão	operação ou marca

Fig. 2

O primeiro parâmetro é o èndereço de um vetor representando uma constante, se encontrado residente no programa do usuário, ou de um vetor representando uma variável, se pertencente a um item de registro [13] quando com qualquer deles se deseja operar. A fim de se poder identificar a procedência deste vetor, estabeleceu-se a condição em que o "bit" mais à esquerda do endereço deve ser ligado quando houver caso de variável. A necessidade de o sistema conhecer a origem do vetor tem como finalidade, em caso de erro de conversão de dado em uma constante, haver interrupção do programa. Os detalhes estão apresentados na descrição do módulo de erro.

Uma outra posição de memória subsequente é reservada para o que se denominou de comprimento. Tem a finalidade de informar ao módulo de conversão o tamanho do item de registro quando se tratar de vetores representando um dado variável. Para o caso de vetores representando um dado constante residente no programa do usuário o valor do comprimento deverá ser zero se ele tiver o formato padrão de IV - 5. É uma questão de superabundância de informação pois para a supracitada questão o tamanho do vetor é indicado na sua posição de memória precedente.

O mesmo acontece com os parâmetros booleanos quer eles pertençam ou não a item de registro. A definição da representação de parâmetro será tratada mais adiante.

Segue-se então o parâmetro de conversão. É neste onde o usuário indica qual rotina deverá ser chamada para converter os dados constantes ou variáveis da sua base de representação para os padrões das rotinas de operação. Por economia de memória determinou-se uma convenção com o intuito de indicar neste parâmetro o tipo de dado - se numérico, booleano ou caracter. Levando-se em consideração uma posição de memória com 8 "bits", tem-se as seguintes representações :

Variáveis ou constantes numéricas

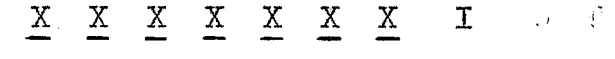


Fig.3

Variáveis ou constantes booleanas

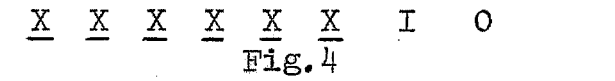


Fig.4

Variáveis ou constantes de caracter

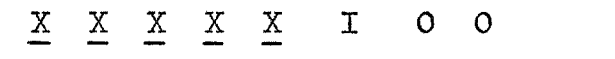


Fig. 5

A exemplo de tudo que se definiu neste trabalho, foram prèviamente pensadas e estudadas as suas melhores vantagens. A padronização dada acima visa economizar memória em termos de programação do sistema em linguagem da 8008. Sendo assim, deter-

minado o tipo de conversão através de instruções de rotação no acumulador, tem-se de imediato o número da rotina de conversão a ser chamada através dos X's (não importa) mostrados nas figuras 3,4 e 5. Como se pode observar o número de rotinas de conversão que o usuário pode chamar é suficiente para qualquer tipo de trabalho - cerca de 31 rotinas diferentes para a condição mais crítica no caso de conversão de caracteres conforme comprovado pela figura 5.

Se o usuário desejar implantar uma nova rotina de conversão, deverá escolher para a sua identificação, um dos padrões apresentados nas figuras 3,4 ou 5, conforme o tipo de dado a ser convertido. Em seguida substituir os X's por um valor numérico decimal em binário subsequente àquela última rotina já existente. Outras informações encontram-se no capítulo sobre módulos

As colunas da tabela de restrições são completadas por mais uma posição de memória denominada de operação ou marca.

Como no caso anterior, convencionou-se na mesma posição de memória a prioridade da operação. Assim tem prioridade 1 os operadores booleanos.

X X X X X X X I

Fig. 6

São as seguintes operações booleanas de dois operadores A e B dada por Dietmeyer [10].

Harrison [14] trata o assunto com profundidade.

	A =	0	0	1	1
	B =	0	1	0	1
0		0	0	0	0
	et ou and	0	0	0	1
	inibição	0	0	1	0
A		0	0	1	1
	inibição	0	1	0	0
B		0	1	0	1
	ou exclusivo	0	1	1	0
	ou inclusivo	0	1	1	1
	nor	1	0	0	0
	nor exclusivo	1	0	0	1
	not	1	0	1	0
	implay	1	0	1	1
	not	1	1	0	0
	implay	1	1	0	1
	nand	1	1	1	0
1		1	1	1	1

Fig. 7

As linhas em branco é porque não existe denominação específica para aquela função booleana. De acordo com essa tabela tem-se as respectivas representações das operações booleanas, fazendo-se variar de 0 a 15 sob a forma decimal binária e

substituir as 7 posições dos X's na figura 6.

Os operadores de comparação lógica numérica ou de caracteres têm prioridades 2 conforme figura 8

X	X	X	X	X	X	I	0

Fig. 8

Os operadores de comparação estão assim representados de acordo com o quadro da figura abaixo :

0	0	0	0	0	0	I	0	menor
0	0	0	0	0	I	I	0	menor ou igual
0	0	0	0	I	0	I	0	igual
0	0	0	0	I	I	I	0	diferente
0	0	0	I	0	0	I	0	maior ou igual
0	0	0	I	0	I	I	0	maior

Fig. 9

Segue-se com prioridade 3 a soma e subtração

0	0	0	0	0	I	0	0	soma
0	0	0	0	I	I	0	0	subtração

Fig. 10

Finalmente multiplicação e divisão com prioridade 4

0	0	0	0	I	0	0	0	multiplicação
0	0	0	I	I	0	0	0	divisão

Fig. 11

Com a mesma prioridade ainda é definida a operação de divisão por 10 ou simplesmente truncamento, como se segue

0 0 I 0 I 0 0 0 divisão ou truncamento

Fig. 12

Os motivos para esta definição estão descritos no próximo capítulo.

É possível notar a versatilidade desta definição que proporciona operações mais sofisticadas, necessárias a um usuário desejoso de fazer operações, tais como potenciação, radiciação, etc.

O formato da figura 2 se repetirá tantas vezes quantas forem necessárias, isto é, quantas forem o número de restrições. Na última posição de memória destinada ao operador de restrição condicionado pelo usuário, deverá conter uma marca, para avisar ao sistema de recuperação de informação o término da tabela de restrição.

Com referência ainda sobre a representação de operadores foi sugerida uma outra representação :

0 0 0 0 P P P P

Fig. 13

onde 0 é a operação e P sua respectiva prioridade. Este modo é simples e tem a vantagem sobre o anterior de o número de operadores possíveis ser constante quando se aumenta de prioridade. Vale salientar que esta padronização não foi adotada porque se

gasta mais memória para a sua implantação. O excesso atinge cerca de 10% ao se programar todo o sistema no terminal inteligente em a padronização estabelecida anteriormente.

IV. MÓDULOS

4.1 - Rotina "FILE"

Esta rotina além de fazer a gerência de todo o sistema de recuperação de informação proposto, executa o algoritmo principal através do uso de duas pilhas: uma para códigos de conversão, dos operadores e outra para endereços. Uma das pilhas tem duas entradas destinadas à conversão de onde se encontra o tipo de operação e outra entrada é destinada ao operador que depois de decodificado se obtém a operação e prioridade da mesma, conforme descrito anteriormente. A outra pilha, com uma única entrada, destina-se a armazenar endereços dos vetores na área de trabalho.

O processo de entrada e saída nas pilhas varia de acordo com a prioridade dos operadores da tabela de restrições. À medida que os ítems da coluna de operadores da tabela de restrições são percorridos, os endereços dos operandos, operadores e conversão entrarão nas pilhas se a prioridade da operação verificada é maior que a última prioridade de operação entrada na pilha de respectivo nome. Assim, em caso contrário, os dois últimos operandos são executados através do último operador da pilha de operação. As duas pilhas diminuem de um nível de entrada e o resultado fica sendo endereçado pela penúltima posição da pilha de endereços. O processo se repete interativamente até que uma marca de prioridade mais baixa que todas as operções é encontrada. Esta marca é feita nas pilhas como a primeira entrada executada nas mesmas. A partir daí, tem-se começado o algoritmo

propriamente dito pelo manuseio sucessivo das pilhas

Cada pilha possui espaço suficiente para doze entradas consecutivas, muito embora só haja necessidade de seis, face às prioridades implantadas: operadores "booleanos", operadores de comparação de caracteres, operadores de soma ou subtração, de multiplicação ou de divisão, marca de início ou fim da pilha. As posições excedentes, reservadas para as pilhas, visam a criação e implementação de outros operadores, além dos citados, com maior prioridade sem exigir a preocupação do usuário com essas pilhas. Por exemplo: potenciação, radiciação, etc.

De modo formal, o número de posições das pilhas é determinado pelo seguinte: o alfabeto definido pelos operadores acima descritos são em número de seis elementos diferentes. Na condição mais crítica, ter-se-ão preenchidas cinco posições da pilha com os respectivos elementos e a próxima entrada será de necessidade obrigatória a um elemento do conjunto, idêntico a um dos já definidos [16]. A sequência é 5-distinguível [1]. Por conseguinte, a sua prioridade será de modo muito estrito igual ao seu antecessor ou menor do que ele, condição exigida para ser executada pelo menos a operação do operador anterior e as pilhas baixarem de um nível.

Não é tratado até aqui o problema de expressões com parênteses, podendo o mesmo ser encontrado no capítulo sôbre Restrições.

Residente na rotina de gerência é encontrado o ende-

reço da área de trabalho, onde nesta são copiados os vetores a serem processados de acordo com a tabela de restrições. A finalidade desta é tornar o sistema "serialmente re-usável", pois tem-se em vista que as rotinas de operação destroem o segundo operando para colocar o resultado final, por questões inerentes ao próprio algoritmo de operação.

O algoritmo da rotina "FILE" encontra-se no apêndice 1 e está descrito na linguagem dada em (I-3).

Sendo n o número de resgistros do arquivo e m o número de restrições determinadas pelo usuário, o tempo de execução do algoritmo é da ordem de n.m. Detalhes sôbre a análise de um algoritmo são encontrados em Knuth [21-22].

As rotinas controladas por este módulo apresentam-se pela seguinte ordem de chamada:

- FITA K7
- VÍDEO
- PRPRCS
- MSTAKE
- Conversão
- Operação

4.2- Rotinas FITA K7 e VÍDEO

Estes programas já estão prontos e depurados.Têm como objetivo todos os controles destes periféricos e estão descritos no Manual do Usuário do S.O.S. [25]. As unidades de "cassette" são usadas para guardar o arquivo de origem enquanto a outra unidade é reservada para o arquivo de destino gerado.

A unidade de vídeo é usada exclusivamente para comunicar ao operador do terminal inteligente, advertências e eventuais erros durante o processamento de qualquer módulo do sistema quando se faz uso da rotina descritiva. Quando é usada a rotina simplificada esta utiliza as luzes.§

§ Veja-se o ítem IV-4 adiante para mais detalhes

4.3 - Rotina PRPRCS

Trata-se de uma rotina de consistência para pré-processamento da tabela de restrições, construida no programa do usuário que verifica se a mesma está construida de forma correta. Faz parte da rotina uma matriz de precedência de operação [12] através da qual é verificada a viabilidade da cálculos dos operandos.

Por meio da precedência de operações é que se constata que a tabela de restrições deve conter o número mínimo de dois operandos onde será obrigatório ser um deles variável, isto é, ser um ítem de registro [13].

É verificada ainda a existência do tipo, da prioridade do operando e da operação respectivamente e a existência da rotina de conversão. Qualquer irregularidade detectada pela violação de quaisquer das condições expostas acima implicará na suspensão da execução do programa do usuário pela rotina de erro do sistema. §

O programa deste algoritmo não foi implantado para diminuir a mão de obra de programação do sistema. Isto implica em não se poder cometer erros na construção da tabela de restrições.

§ Ver algoritmo no apêndice.

4.4 - Rotina MSTAKE

São apresentadas duas versões à escolha do usuário. Uma versão discriminadora e outra concisa. A primeira informa de modo descritivo, pelo vídeo, as causas das advertências e erros com consequênte interrupção pelo sistema. Já a versão concisa não informa mais que um número indicado nas luzes a ser consultado em um manual onde este descreve o acontecimento certificando ainda se se trata de advertência ou erro através de um sinal sonoro.

Os motivos das duas versões são:

a - dar ao usuário de um terminal com capacidade de memória maior que 4 K-"bytes", uma condição cômoda e confortável - versão 1;

b - dar a mesma eficiência de trabalho e informação aquele usuário cujo terminal só dispõe de 4 K-"bytes" de memória - versão 2

A versão 1 ocupa 1.200 unidades de memória enquanto a versão 2 necessita de apenas 20.

Os algoritmos para as duas versões são iguais com a única diferença de que enquanto a descriminadora envia mensagem pelo vídeo, a rotina simplificada o faz pelas luzes.

4.5 - Rotinas de conversão

Depois de pré-processados, os parâmetros são submetidos às rotinas de conversão correspondentes conforme sejam operandos "booleanos", numéricos ou de caracteres. As conversões são feitas para os seguintes padrões pré-estabelecidos:

- Os parâmetros "booleanos" são convertidos de qualquer conversão para todos os "bits" de um byte ligados se a condição é verdadeira e, em caso contrário, "mutatis mutandis", todos os "bits" deverão estar ligados quando a condição for falsa. Para se representar uma variável "booleana" é necessário apenas uma posição de memória.

Os parâmetros numéricos devem ser transformados para a base decimal em cuja representação, o primeiro "byte" é destinado para o seu comprimento, reservando-se o "bit" de mais alta ordem para indicar o sinal deste parâmetro. Assim, "bit" ligado significa parâmetro numérico negativo e desligado, quando se tratar de parâmetro numérico positivo. Daí, então, deve-se seguir tantas posições de memória quantas forem o tamanho prescrito e cada uma delas deve conter um único número entre zero e nove. Desse modo pode ser representado qualquer número na base decimal. Ver maiores detalhes no próximo ítem onde será dado informações sobre cada uma das rotinas numéricas e em Harrison [14].

Quanto aos parâmetros de caracteres ou "string" valem as mesmas considerações para a primeira posição, descrita para os numéricos, com a exceção de não existir "bit" de sinal. De

vem seguir-se, sobretudo, tantos caracteres em ASCII por posição de memória quantas forem o comprimento estabelecido pela primeira posição.

Seja qual for o sistema de representação de qualquer tipo "booleanos", numéricos, ou de caracteres, eles devem ser convertidos para os padrões descritos acima.

Um atributo muito importante realizado pelas rotinas de conversão é a de mover para a área de trabalho os dígitos após a conversão. Evidentemente que, se os parâmetros, já satisfizerem os padrões, sòmente existirá a necessidade de copiar os vetores para área de trabalho.

É bom lembrar que os padrões de conversão estabelecidos acima não são rígidos e podem ser modificados ao bel-prazer do usuário. Os padrões constituem apenas um ponto de partida da definição do sistema original. Quando se falou em III-2 que o sistema opera sob uma base de padrões pré-estabelecidos, quis-se dizer apenas o caminho menos trabalhoso a ser seguido por um usuário que deseje trabalhar com o sistema aqui desenvolvido onde os seus dados estejam representados sob formato diferente. Por exemplo se um usuário tem um arquivo com dados numéricos na base hexadecimal poderá operá-los de dois modos. O primeiro é fazer uma rotina de conversão para mover os números para a área de trabalho e construir as rotinas de operação numérica hexadecimal. Na segunda opção ele necessitará fazer apenas uma rotina de conversão da base hexadecimal [14] e mover também estes dígitos para área de trabalho. A princípio do que foi dito, não resta

dúvidas quanto a escolha da segunda opção pois, custa menos esforço.

As rotinas de conversão devolvem ao programa de controle uma indicação de conversão feita, se normal ou irregular. Contudo, compete à rotina de erros interromper a execução do sistema ou simplesmente advertir o usuário se a irregularidade detectada procede de um vetor representando uma constante ou variável respectivamente.

Por questões práticas é apresentado no apêndice um único algoritmo de cada tipo de conversão. De qualquer modo a conversão a ser feita é uma função do arquivo e convenções do usuário.

4.6 - Rotinas de operação

O último tratamento que os vetores operando sofrem dizem respeito às operações de cálculo propriamente ditas. Da mesma forma dividida para as rotinas de conversões , estas também estão divididas em operações "booleanas", de caracteres e numéricas. As operações "booleanas" realizadas são todas aquelas mostradas no quadro da figura 7 capítulo III-6. Os algorítmos referentes ao cálculo de "Boole" não serão aqui apresentados visto serem, por demais, simplórios.

No caso particular da UCP 8008, um vetor "booleano" é representado por uma única posição de memória e terá valor FF_{16} se verdadeiro e 0 se falso, conforme descritos no item anterior. Essa rotina devolve um indicador de processamento normal ou anormal se o operador é desconhecido. Este tipo de erro é fatal, implicando na interrupção de processamento pela rotina MSTAKE.

Um vetor para ser operado numèricamente já foi descrito de forma geral no item anterior. Há entretanto uma exceção : a rotina de divisão por dez. Ela tem como finalidade principal, o controle pelo usuário do ponto ou vírgula decimal. Assim, todas as rotinas podem operar com quaisquer números reais tratando-os como se fossem inteiros. Este fato deu mais conforto para o autor do trabalho permitindo-o operar com números não formatados. Esta tarefa fica a cargo do usuário no momento da edição. Note o leitor que isto não é mau para o usuário, pois

o condiciona de estar a par ou, ao menos, ter noção dos resultados. Saiba ainda que cuidados semelhantes são necessários para se evitarem os sérios problemas de "overflow" e "underflow" quando se trabalha com números, com ponto fixo ou flutuante. Foi uma das experiências adqueridas pelo autor ao concluir o seu trabalho de Química sobre curva de calibração [27]. Isso acontece independentemente da marca ou modelo do computador. O primeiro operando da rotina de divisão por dez é um operando numérico decimal nos padrões já descritos. O segundo operando é uma única posição de memória e informa a potência de dez que o primeiro operando deve ser truncado. Logo este expoente, no terminal inteligente, pode alcançar o valor máximo de 127. Por este motivo de exceção, número não representa tamanho do vetor, é necessário uma rotina de conversão exclusivamente para o segundo operando. Esta rotina devolve um indicador de erro se a potência de dez é maior que o comprimento do primeiro operando.

Não se adiantará mais que, as rotinas de adição subtração, multiplicação e divisão operam com algoritmos que trabalham com os vetores numéricos de forma esdrúxula, da esquerda para a direita. Detalhes e fluxograma dessas rotinas podem ser encontrados no manual de referência de programas do sistema comercial para o 1130 da IBM [17].

Com exceção da operação de divisão, todas as operações descritas neste capítulo estão implantadas.

Para terminar este item falta falar sobre as rotinas

de operações com caracteres. Elas são exclusivamente de operação e têm os operadores iguais aos numéricos descritos e apresentados no quadro da figura 9. A diferênça básica entre as rotinas de comparação numérica e as de caracteres é que a primeira não leva em consideração zeros ou brancos à esquerda do número. Já as rotinas de comparação de caracter os leva em consideração. Como consequência disto dois vetores de caracteres só serão iguais se tiverem além de comprimento, conteúdos também iguais.

V. RESTRIÇÕES

Neste capítulo sobre restrições serão apresentadas as principais limitações de algoritmo e programação inerentes ao sistema de recuperação de informação. Algumas das dificuldades descritas, entretanto podem ser eliminadas e vistas no próximo capítulo.

5.1 - Limitação de algoritmo

A fim de melhor esclarecer o raciocínio seguinte, definir-se-ão perguntas de relação e perguntas de nome de condição [19]. Admita-se um arquivo de cadastro dos funcionários de uma empresa. Um conjunto de perguntas é de relação quando se trata de comparar um conjunto de dados com um outro especificado. Assim, temos como exemplo de perguntas de relação:

- quais os funcionários maiores de 18 anos?
- quais os funcionários que ganham 3 salários mínimos?
- quais os funcionários que não possuem dependentes?

Como perguntas de nome de condição citem-se os seguintes exemplos :

- qual o funcionário mais velho da empresa?
- quais os funcionários que mais ganham na empresa?
- quais os funcionários que possuem maior número de dependentes?

No primeiro conjunto a resposta parte de uma determinada referência que no caso é 18 anos, três salários mínimos e zero dependentes; daí a razão de se chamar perguntas de relação.

No segundo caso não é fornecida uma referência como no caso anterior. Um detalhe importante a se notar é que, admitindo-se uma empresa com funcionários, para o primeiro con-

junto de perguntas chamadas de relação poderá ou não haver respostas. Com referência aos exemplos dados, basta os funcionários serem menores de ou iguais a 18 anos, não haver funcionários ganhando três salários mínimos etc., para não existir respostas. Entretanto, para o conjunto de perguntas de nome de relação a resposta sempre haverá. Ainda que no caso mais extremo de a empresa possuir apenas um único funcionário.

Uma primeira restrição de algoritmo do sistema de recuperação de informação desenvolvido para o terminal inteligente, é não aceitar perguntas de nome de condição.

Uma outra restrição do sistema é não aceitar perguntas quantitativas isto é, só aceitar perguntas qualitativas. Isto implica em jamais poder fazer perguntas com o pronome indefinido quantos.

O motivo que levaram a estas restrições é não se poder representar de modo simples, as perguntas sob a forma da tabela definida no capítulo III.

5.2 - Não admite parênteses

Para o sistema admitir parênteses nas expressões construidas através da tabela de restrições, seria necessário inicialmente limitar o número de existência dos mesmos, pois, é estudado em teoria dos Autômatos que não há máquina finita capaz de reconhecer expressões com número ilimitado de parênteses [24].

Mesmo determinando-se o limite do máximo de parênteses, ainda se tornaria um pouco confuso para o usuário representá-los sob a forma de tabela. O objetivo aqui é tornar o sistema mais simples para o usuário. E esta simplicidade é conseguida se antes o usuário eliminar algebricamente os parênteses da expressão.

Apesar disso é dado no próximo capítulo uma maneira de se trabalhar diretamente com expressões contendo parênteses.

5.3 - Limitação de programação

A UCP 8008 é projetada para trabalhar com uma pilha com sete posições destinadas a chamadas de rotinas, número de níveis de subrotina.

O sistema está programado para ser usado no máximo como chamada de um subprograma do usuário. Como se pode observar, fica o usuário restringido a apenas dois níveis. Este inconveniente pode ser minorado colocando-se à disposição mais níveis de chamada, assunto que será tratado mais adiante.

Pode ser considerada como uma outra restrição de programação o grande número de variáveis externas. Este modo de transmitir parâmetros para rotinas julgou-se melhor do que através do endereçamento de uma lista de parâmetros por registros duplos.

Assim sendo, será exigido menos cuidado de análise do sistema para uma possível modificação em quaisquer dos seus módulos. Outra vantagem é naturalmente, a economia dos dois registros que endereçam a lista de parâmetros.

VI . SUGESTÕES

6.1 - Redução do nível da pilha de chamada das rotinas

A primeira modificação a ser feita diz respeito ao local de chamada de cada rotina. As instruções de chamadas devem ser trocadas por uma instrução que transmita através de registros o endereço de retorno da rotina seguida de um comando de desvio para a mesma.

Necessário se faz ainda que as rotinas sejam modificadas da seguinte forma :

- As primeiras instruções devem guardar o endereço de retorno no local do operando de uma instrução desvio;
- Uma instrução de desvio contendo o endêreço de retorno ao local de chamada, substitui o tradicional comando de retorno.

Contudo há os seguintes inconvenientes :

1 - é pouco elegante modificar o programa;
2 - aumenta de 4 "bytes" a chamada de rotina e esta por sua vez é aumentada de 9 "bytes";
3 - a rotina chamada tem de ser de nível maior que dois ou não adiantará por causa da RST INCHL;
4 - gasta mais dois registros para transmitir o endereço de retorno.

Apesar disto poderia ser usado em algumas rotinas. Estas modificações razoavelmente grande dá ao usuário três níveis de chamada de rotina diminuindo o nível total do sistema para três. Naturalmente, já foi dito, este ganho vem em detrimento da memória útil disponível do usuário por causa de maior gasto de memória do sistema.

6.2 - Solução para os Parênteses

Sabe-se do capítulo anterior que para tornar o algorítmo computável deve ser estabelecido o número máximo de parênteses admitido pelo sistema [24]. Deverá ser reservada uma posição de memória indicando o número de parênteses necessários, menores que ou iguais ao limite determinado, ao lado direito de cada linha da tabela de restrições mostrada na figura 2. Ainda mais, nesta mesma posição, o "bit" D7 é reservado para indicar o tipo de parêntese. Por exemplo, "bit" D7 desligado indica "abre parêntese". Em caso contrário significará "fecha parêntese".

Para economizar-se memória, no caso de ausência de parênteses o "bit" D6 do "byte" de mais alta ordem de posição do endereço deverá estar desligado. Havendo parênteses deverá estar ligado. Isto condiciona o usuário a só utilizar maior memória quando houver parênteses. Conclui-se fàcilmente que a tabela de restrições deixa de ter o formato retangular e passa a apresentar-se de modo irregular, de acordo com a existência ou não desses delimitadores.

Há dois algorítmos para o tratamento de operações de expressões com parênteses. O primeiro deles utiliza uma pilha para operandos e outra para operadores, tal como descrito em capítulo anterior.

A cada vez que surge um "abre parêntese", então todas as prioridades dos operadores, a partir daí, são somadas à prioridade do operador de maior prioridade; seguindo-se os mes-

mos passos do algoritmo descrito no capítulo 4, até que a cada " fecha parêntese" encontrado, esta soma deixa de existir. O objetivo disso é, como normalmente ocorre com o leitor quando está resolvendo expressões algébricas, dar prioridade de resolução a todas as operações contidas dentro do parêntese mais interno.

Já Peter Wagner [28] apresenta um outro algorítmo proposto por Randell e Russell [26] : a exemplo do exposto acima também há duas pilhas para entrada de operadores e operandos. Agora a manipulação das pilhas é feita assim :

- os operandos são transmitidos diretamente para a saída;
- o parêntese da esquerda é colocado na pilha de operadores com prioridade zero;
- o parêntese da direita causa a retirada dos operadores da pilha até que o parêntese da esquerda é encontrado. Este é então removido;
- cada operador restante tem uma prioridade. Assim, em cada operador é encontrado sua prioridade, confrontado com a do operador do topo da pilha; os operadores são removidos da pilha até um operador com prioridade menor que a deste operador em causa é encontrado.

A diferença básica entre os dois algorítmos é que no último caso, os parênteses da esquerda entram na pilha de operadores, enquanto no primeiro algorítmo, ao encontrar um parêntese esquerdista soma, a partir daí, a prioridade dos opera-

dores, a maior prioridade existente entre os operadores, até que um parêntese direitista é encontrado.

6.3 - Outra sugestão

Um usuário do terminal inteligente pode desejar desenvolver um sistema para decodificação de registros. Por exemplo,poderá desejar saber de um arquivo de estoque de mercadoria : qual o estoque mínimo de uma mercadoria X ? O leitor pode comprovar aqui que o usuário está fazendo uma pergunta de nome de condição e, como já foi dito, o sistema no seu estado atual não responde a estes tipos de perguntas. Entretanto, com algumas modificações acrescentadas ao sistema o usuário minorará o seu trabalho evitando a definição, desenvolvimento e execução de um novo sistema.

O usuário deverá substituir o módulo de gravação em fita "cassette" por um módulo de decodificação de registros. Observe você, isto não modifica as ideias entre o sistema original e o novo. No primeiro caso, após a tabela de restrição ter chegado ao fim, o registro é gravado, em caso verdadeiro; em caso contrário, passa-se ao registro subsequente. Já no novo sistema, quando o registro da mercadoria X for encontrado, o módulo de decodificação deverá buscar no endereço dado pelo usuário - campo de estoque mínimo da referida mercadoria - o valor do seu estoque mínimo e transmiti-lo através de um periférico de saída conveniente : o vídeo, por exemplo. No caso de a mercadoria X não ser encontrada, o sistema original envia mensagem de fim de processamento.

É oportuno lembrar que dentre as sugestões aqui apresentadas, esta exige maior volume de trabalho, pois, ter-se-á de se definir um módulo de decodificação.

VII. CONCLUSÕES

7.1 - Desempenho do sistema

Neste item será mostrada a eficiência do sistema de recuperação de informação para microprocessadores com um exemplo prático. Os dados apresentados tem um erro por excesso para todas as condições em sua fase mais crítica por motivo de segurança.

O sistema total ocupa 3,80 K "bytes" de memória quando usado com a rotina de erro simplificada, deixando aproximadamente 5% disponível para um usuário de um terminal com mínimo de 4,00 K"bytes". Esta memória disponível satisfaz a pequenas aplicações comerciais e industriais considerando-se que :

- o programa teste do sistema ocupa apenas 0,15 K "bytes" isto é, 3/4 de memória disponível das quais 70% estão reservadas para área de trabalho e leitura de registros;
- no trabalho desenvolvido e no programa teste, não se utilizou as facilidades de "overlay" oferecidas pelo terminal inteligente. O usuário deve usar este recurso para um programa mais elaborado. É possível diminuir de 1,50% a memória ocupada pelo sistema fazendo-se maior uso da rotina INCHL para endereçamento de memória. Isto vem em detrimento do tempo gasto e documentação do programa.

O programa teste opera com a seguinte expressão matemática :

$$I3 + I4 > C;$$

onde I3 e I4 são respectivamente o terceiro e quarto items de cada registro de um arquivo exemplo e C é uma constante residente no programa teste com valor 2800.‡

O sistema satisfaz aos objetivos mostrados em I-2, tornando-o vital e indispensável ao terminal inteligente quando utilizado como um simples microcomputador. Isto não impede de o sistema ser chamado em um terminal inteligente ligado a um computador de grande porte.

‡ Ver exemplos com respetivos resultados no Apendice

7.2 - Conclusão final

Estudiosos como Codd e Boyce preferem usar o cálculo relacional para consulta em arquivo, como melhor forma do que estrutura em árvore e esta melhor do que a sequencial. Eles trabalham com a forma normalizada pelos seguintes motivos [2,8] :

a - ser possível dispor em quadros, qualquer base de dados;

b - obter uma poderosa capacidade de recuperação, por meio de um simples conjunto de relação, melhor do que obtido de outro modo;

c - livrar o conjunto de relações das indesejáveis dependências de inserção, atualização e retirada;

d - reduzir a necessidade de reestruturar o conjunto de relações como novo tipo de dado introduzido e assim, aumentar a capacidade de aplicação do programa;

e - fazer o modelo relacional mais informativo para o usuário;

f - fazer o conjunto de relações indiferente a perguntas estatísticas onde estas estatísticas são obrigadas a mudar à medida que o tempo passa.

Codd acrescenta ainda que o quadro de uma tabela é normalmente entendido como uma disposição retangular com as seguintes propriedades :

P1 - ser homogénea em colunas - em outras palavras,

em qualquer coluna selecionada, todos os items são da mesma espécie, ao passo que em diferentes colunas não necessitam ser homogêneas;

P2 - cada item é um simples número ou um vetor de caracteres;

P3 - todas as linhas de uma tabela devem ser distintas;

P4 - a ordem das linhas na tabela é inconsequente;

P5 - as colunas de uma tabela são designadas por nomes diferentes e a sua ordem é inconsequente.

É possível observar que o trabalho de recuperação de informação para microcomputadores não se enquadra no modelo relacional normalizado de Codd [8] pois, o sistema desenvolvido ignora o fato de haver linhas na tabela em conflito. Isto vai de encontro com P3. Por outro lado, não há módulo no sistema que faça tratamento de relações de base de dados, como diz Heath [15]. Por esta razão o autor não levou em consideração até aqui os problemas de violação desses princípios apresentados. Para este fim Heath sugere uma interface entre o programa do usuário e o sistema de gerência de dados chamado de "third normal form" TNF. É importante, a exemplo do que foi apresentado no capítulo III, que a ordem das colunas na tabela fica a cargo do prévio conhecimento do arquivo a ser utilizado pelo usuário. Isto vai de encontro com P5.

A P Ê N D I C E

Algorítmo da rotina FILE

```
FILE (tabela de restrições, área de leitura em K7,
chave da MSTAKE)
BEGIN
Reenrola fitas de origem e destino; contador da área
de trabalho : = 0; ler registro na fita de origem;
LOOP UNTIL  "eof" da fita de origem OR contador área
da pilha > 12 OR erro de conversão de constante:
Inicia as pilhas e variáveis : = 0; (prioridade topo
da pilha, prioridade anterior) contador área da pilha:
= 1; contador da tabela : = 0;
DO
BEGIN
Incrementa contador área da pilha;
IF contador da área da pilha ┌ > 12
THEN
Incrementa contador da tabela;
Prioridade anterior : = prioridade topo da pilha;
Prioridade topo da pilha : =
        IF prioridade da tabela = "flag" ligado
        THEN
        0
        ELSE
        Prioridade da tabela ;
```

```
FI
Conversão topo da pilha : = conversão da tabela;
Operador anterior : = operador de tabela;
Coloca nas pilhas endereço, operador, prioridade e
tipo da tabela;
CASE tipo topo da pilha OF
BEGIN
CALL NCHn; (converte numérico)
CALL SCHn; (converte caracteres)
CALL BCHn; (converte booleano)
END
    IF não erro de conversão em constante∧variável
    THEN
        WHILE prioridade operador ¬> prioridade
        operador topo da pilha ∧ contador de área
        de trabalho > 2  DO
        BEGIN
        CASE tipo anterior OF
            BEGIN
            CALL NMERIC; (operação numérica)
            CAAL BOLEAN; (operação"booleana")
            CALL STRING; (operação de caracteres)
            END
        Decrementa contador área da pilha;
        Prioridade anterior : = topo da pilha - 2 ;
        Topo da pilha-1:=prioridade topo da pilha;
        END
```

FI

FI

END

UNTIL prioridade topo da pilha ≠ /00 ∧ contador da área da pilha > 12 ∧ não erro de conversão em constante ∧ variável

IF último resultado da área de trabalho verdadeiro ∧ contador de área de pilha ≤ 12 ∧ não erro de conversão em constante ∧ variável

THEN

Grava registro na fita de destino;

IF fim de fita de destino

THEN

Apaga registro da fita de destino;

Pede nova fita de destino;

Reenrola fita de destino;

Grava registro na fita de destino;

FI

FI

IF contador da área da pilha ≤ 12 ∧ não erro de conversão em constante

THEN

Ler registro na fita de origem; (testa fim de volume, pede outro volume, reenrola fita de origem e ler registro na fita de origem)

FI

REPEAT;

```
"eof" de fita de origem => rotula fita de destino
com "eof";
contador da área da pilha > 12 => CALL MSTAKE;
erro de conversão de constante => CALL MSTAKE;
FI
END
```

n - número de rotinas de conversão de cada tipo existente.

O programa é interrompido em quaisquer destas 3 últimas ocorrências. Note-se ainda que um novo registro é lido quando há erro de conversão em variável ≡ vetor representando item de registro.

Algoritmo da rotina PRPRCS

```
PRPRCS (tabela de restrições, chave de erro)
BEGIN
      I ← 1;
Desliga chave; decodifica operador (I) da tabela;
LOOP UNTIL marca de fim de tabela de restrições
OR tipo inexistente OR prioridade inexistente OR
precedência de operador inválida :
Decodifica operador (I + 1) da tabela;
IF tipo existe
THEN
     IF prioridade existe
     THEN
     Consulta matriz de precedência;
            IF precedência certa
            THEN
            I ← I + 1;
            ELSE
            Precedência inválida; liga chave;
            FI
     ELSE
     Prioridade inexistente; liga chave;
     FI
ELSE
Tipo inexistente; liga chave;
FI
REPEAT;
```

```
THEN
marca fim de tabela de restrições => chave de
erro <- chave;
Tipo inexistente          => chave de erro <- chave;
Prioridade inexistente    => chave de erro <- chave;
Precedência inválida      => chave de erro <- chave;
FI
END
```

Estas três últimas ocorrências, faz a rotina retornar com um código que interrompe o processamento do sistema através da rotina MSTAKE.

Algoritmo da rotina de conversão NCHØ

```
NCHØ (comprimento do vetor, área de trabalho,
contador da área de trabalho, "flag")
BEGIN
Contador da área de trabalho ← contador da área
de trabalho + comprimento do vetor;
IF contador da área de trabalho ⌐> área de traba-
lho
THEN
Identificação do tipo de vetor pelo "flag" cons-
tante ou variável;
DO
BEGIN
"Scan" no vetor numérico;
     IF dígito > 9
     THEN
          IF "flag" ligado
          THEN
          Erro em variável;
          ELSE
          Erro em constante ;
          FI
          Retorna;
     ELSE
     Converte ASCII  em decimal;
     Move dígito para área de trabalho;
     Decrementa contador vetor numérico;
```

```
    FI
END
UNTIL contador vetor numérico = 0 ∧ não erro
ELSE
"overflow" da área de trabalho;
FI
END
```

Algorítmo da rotina de conversão BCHØ

BCHØ (área de trabalho, contador da área de trabalho, "flag")

BEGIN

Contador da área de trabalho ← contador da área de trabalho + 1;

IF contador da área de trabalho ⌐ > área de trabalho

THEN

Identificação do tipo de vetor pelo "flag"; (constante ou variável)

"Scan" no parâmetro "booleano";

IF parâmetro "booleano"¬ = 'X' | ¬ = 'b'

THEN

IF "flag" ligado;

THEN

Erro em variável;

ELSE

Erro em constante;

FI

Retorno;

ELSE

IF parâmetro "booleano" = 'x'

THEN

Move /FF para área de trabalho;

ELSE

Move /00 para área de trabalho;

```
            FI
        FI
ELSE
"Overflow" na área de trabalho;
FI
END
```

Algorítmo de rotina de conversão SCHØ

SCHØ (comprimento do vetor, área de trabalho, contador da área de trabalho)

BEGIN

Contador da área de trabalho ← contador da área de trabalho + comprimento do vetor;

IF contador da área de trabalho ¬ > área de trabalho

THEN

Identificação do tipo do vetor pelo "flag"; (constante ou variável)

DO

BEGIN

"Scan" no vetor de caracteres;

Move caracter para área de trabalho;

Decrementa contador vetor de caracter;

END

UNTIL contador vetor de caracter = 0;

ELSE

"Overflow" na área de trabalho;

FI

END

Algaritmo da rotina de interface "booleana"

```
BOLEAN (parâmetros, operador booleano, apontador da
área de trabalho, contador da área de trabalho)
BEGIN
CASE, operador "booleano" OF
BEGIN
Operação ZERO;
Operação"AND";
Operação "INHIBIT";
Operação A;
Operação "INHIBIT";
Operação B;
Operação "EXCLUSIVE OR";
Operação "INCLUSIVE OR;
Operação "NOR";
Operação "EXCLUSIVE OR";
Operação "NOT"
Operação "IMPLAY";
Operação "NOT";
Operação "IMPLAY";
Operação "NAND"
Operação UM
END
atualiza apontador da área de trabalho;
atualiza contador da área de trabalho:
END
```

Algoritmo da rotina de interface de caracteres

```
STRING (parâmetros, operador de caracter, apontador
da área de trabalho, contador da área de trabalho)
BEGIN
CASE operador de caracter OF
BEGIN
CALL SLESS;      (menor)
CALL SLSQAL;     (menor ou igual)
CALL SEQUAL;     (igual)
CALL SDFFRT;     (diferente ou não igual)
CALL SGRQAL;     (maior ou igual)
CALL SGRETR;     (maior)
END
Atualiza apontador da área de trabalho;
Atualiza contador da área de trabalho;
END
```

Algoritmo da rotina de interface numérica

```
NMERIC (parâmetros, operador numérico, apontador de
área de trabalho, contador da área de trabalho).
BEGIN
CASE operador numérico OF
BEGIN
CALL NLESS;        (menor)
CALL NLSQAL;       (menor ou igual)
CALL NEQUAL;       (igual)
CALL NDFFRT;       (diferente ou não igual)
CALL NGRQAL;       (maior ou igual)
CALL NGRETR;       (maior)
CALL ADD1;         (adição)
CALL SUB1;         (subtração)
CALL MPY1;         (multiplicação)
CALL DIV1;         (divisão)
END
IF não divisão por 10
THEN
Atualiza área de trabalho;
ELSE
CALL DIV10;        (truncar ou divisão por dez)
FI
Atualiza apontador da área de trabalho;
Atualiza contador da área de trabalho;
END
```

Algoritmo exemplo da rotina de comparação numérica

```
NMAIOR (vetor 1º operando, comprimento do 1º operando, vetor 2º operando, comprimento do 2º operando, resultado)
BEGIN
IF sinais dos dois operandos iguais
THEN
WHILE          digito à esquerda do 1º operando=0 DO
BEGIN
Seleciona dígito do 1º operando;
Decrementa comprimento do 1º operando;
END
WHILE dígito à esquerda do 2º operando = 0 DO
BEGIN
Seleciona dígito do 2º operando;
Decrementa comprimento do 2º operando;
END
     IF comprimento dos dois operandos são iguais
     ∧ ≠ 0
     THEN
     WHILE  comprimento do 1º operando ≠ 0 DO
     BEGIN
     Seleciona dígito do 1º e 2º operandos
          IF dígito de 1º operando > dígito de 2º
          operando;
          THEN
```

```
                    IF sinal do 1º operando positivo
                    THEN
                    Resultado verdadeiro; Retorna;
                    ELSE
                    Resultado falso; Retorna;
                    FI
               FI
               IF dígito do 1º operando < dígito do 2º
               operando
               THEN
                    IF sinal do 1º operando positivo
                    THEN
                    Resultado verdadeiro; Retorna;
                    ELSE
                    Resultado falso; Retorna;
                    FI
               FI
          Decrementa comprimento do 1º operando;
          END
          FI
Resultado falso;
Retorna
FI
          IF sinal do 1º operando positivo
          THEN
          Resultado verdadeiro; Retorna;
          ELSE
```

```
                Resultado falso; Retorna;
                FI
            END
```

Esta rotina devolve em Resultado a condição verdadeira /FF se o primeiro operando é maior que o segundo e /00 em caso contrário.

Algorítmo exemplo de rotina de comparação de caracteres

```
SMAIOR (vetor do 1º operando, comprimento do 1º operando, vetor do 2º operando, comprimento do 2º operando, resultado)
BEGIN
IF comprimento dos dois operandos iguais
THEN
WHILE, comprimento do 1º operando ≠ 0 DO
BEGIN
Seleciona dígito do 1º e 2º operandos
     IF dígito do 1º operando > dígito do 2º operando
     THEN
     Resultado verdadeiro; Retorna;
     FI
     IF dígito do 1º operando < dígito do 2º operando
     THEN
     Resultado falso; Retorna;
     FI
Decrementa comprimento do 1º operando;
END
Resultado falso; Retorna;
FI
     IF comprimento do 1º operando > comprimento do 2º operando
```

```
        THEN
        Resultado verdadeiro; Retorna;
        ELSE
        Resultado falso; Retorna;
        FI
END
```

SOS - V1M00 * MONTADOR * DATA 23/06/76

```
          * PROGRAMA TESTE DA ROTINA FILE
          *
          *.....
          *
                    ABS       /870
                    EXT       FILE
                    LAI       /01
                    LRD       K7
                    LRB       WORK
                    CAL       FILE
                    HLT
0050      KSETE     DS        80
          K7        DC        D(KSETE,80)

001E      AREA      DS        30
          WORK      DC        D(AREA,30)

                    DC        D(CTE)              C
                    DC        (0,/01,/16)         >


                    DC        D(KSETE+/8000+17)   I3
                    DC        (5,/01,/04)         +


                    DC        D(KSETE+/8000+22)   I4
                    DC        (5,/01,/00)         FIM DE TABELA


          CTE       DC        (/04,2,8,0,0)



                    END
```

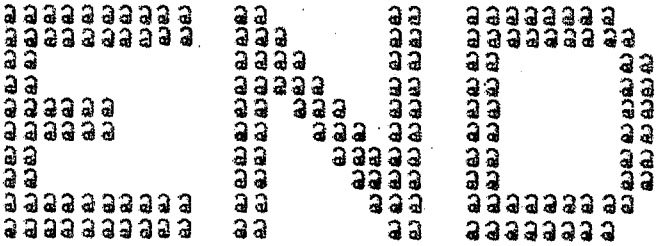
BYTES MONTAGEM BEM SUCEDIDA 0 ADVERTENCIA(S) 0 ERROS(S)

OPCOES EM EFEITO NXREF LIST NTSIM

01/PASS02 3,755/ 3,775 SEG

```
TERMINAL INTELIGENTE  SOS - V1M00          *   LOADER *

        MAPA DE ALOCACAO DA MEMORIA

      NOME       ENDERECO

      TESTE1     0100

      EVIDEO     0245

      EVIDEO     0245


  TEMPO DE EXECUCAO DO LOADER            5  SEC.

***  PA - PARADA EM        +/0100 (+00256)
ENTRADA PELO CONSOLE
MAN
          HLT -   PARDU    EM  IAR - 0149
***  PA - PARADA EM        +/0149 (+00329)
ESCREVA '  *** EXEMPLO COM A ROTINA FILE PARA   C = 2800 ***'
  *** EXEMPLO COM A ROTINA FILE PARA   C = 2800 ***
COP
                     H= 00                  V= 11
  *** ARQUIVO ORIGINAL DE BUSCAS DA ROTINA FILE ***
EDUARDO..160749 02570 0082PENEDOAL
DORIA     230451 04780-2128MACEIOAL
SILVA     101154 00564 0739RECIFEPE
CAMPOS    150762 03239-0135MACEIOAL
SANTOS...180348 01520 0352RIO DE JANEIRO
  *** ARQUIVO GERADO A PARTIR DE    C > I3 + I4 ***
EDUARDO..160749 02570 0082PENEDOAL
DORIA     230451 04780-2128MACEIOAL
SILVA     101154 00564 0739RECIFEPE
SANTOS...180348 01520 0352RIO DE JANEIRO

FIM
```

```
****************** 9758
****************** 9758
****************** 9758
****************** 9758
****************** 9758
****************** 9758
****************** 9758
****************** 9758
****************** 9758
****************** 9758
```



UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
UFRJ

```
          * PROGRAMA TESTE=1 DA ROTINA FILE
          *
          *.....*
          *
                    ABS       /B70
                    EXT       FILE
                    LAI       /01
                    LRD       K7
                    LRB       WORK
                    CAL       FILE
                    HLT
0050      KSETE     DS        80
          K7        DC        D(KSETE,80)

001E      AREA      DS        30
          WORK      DC        D(AREA,30)

                    DC        D(KSETE+/8000+17)   I3
                    DC        (5,/01,/04)         +


                    DC        D(KSETE+/8000+22)   I4
                    DC        (5,/01,/16)         >


                    DC        D(CTE)              C
                    DC        (0,/01,/00)         FIM DE TABELA


          CTE       DC        (/04,2,8,0,0)



                    END
```

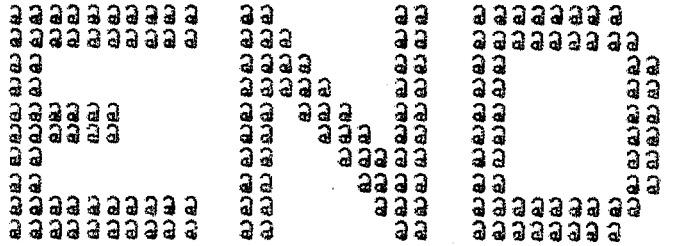
BYTES MONTAGEM BEM SUCEDIDA 0 ADVERTENCIA(S) 0 ERROS(S)

OPCOES EM EFEITO NXREF LIST NTSIM

01/PASSO2 3,360/ 2,535 SEG

```
TERMINAL INTELIGENTE  SOS - V1M00          *   LOADER *

     MAPA DE ALOCACAO DA MEMORIA

NOME       ENDERECO

TESTE1     0100

EVIDEO     0245

EVIDEO     0245


  TEMPO DE EXECUCAO DO LOADER            4  SEC.

***  PA - PARADA EM        +/0100 (+00256)
ENTRADA PELO CONSOLE
MAN
          HLT -   PAROU    EM  IAR - 0149
***  PA - PARADA EM        +/0149 (+00329)
ESC *  *** EXEMPLO-1 COM A ROTINA FILE PARA C = 280) ***'
  *** EXEMPLO-1 COM A ROTINA FILE PARA C = 2800 ***
COP
                    H= 00                V= 11


  *** ARQUIVO ORIGINAL DE BUSCAS DA ROTINA FILE ***
EDUARDO..160749 02570 0082PENEDOAL
DORIA     230451 04780-2128MACEIOAL
SILVA     101154 00564 0739RECIFEPE
CAMPOS    150762 03239-0135MACEIOAL
SANTOS...180348 01520 0352RIO DE JANEIRO
  *** ARQUIVO GERADO A PARTIR DE    I3 + I4 > C ***
CAMPOS    150762 03239-0135MACEIOAL

FIM
```

```
****************** 2729
****************** 2729
****************** 2729
****************** 2729
****************** 2729
****************** 2729
****************** 2729
****************** 2729
****************** 2729
****************** 2729
```



UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
UFRJ

BIBLIOGRAFIA

(1) - AHO, Alfred V., ULLMAN, Jeffrey D. - The Theory of Parsing Translation, and Compiling Vol 1., Englewood Cliffs N.J., Prentice-Hall, 1972, pp 103-132.

(2) - BOYCE, R.F. - Specifying Queries As Relational Expressions "Square", IBM Technical Report, September, 1973.

(3) - BURROUGHS - Algol Reference Manual B6700/B7700 B5000649, Detroit, Michigan, Burroughs Corporation, May 1974, 5 (1-108).

(4) - BURROUGHS - Medium Systems, Reporter System Reference Manual nº 1066693, Detroit, Michigan, Burroughs Corporation, May 1973, 120 p.

(5) - CHANDOR, Anthony - A Dictionary of Computers, Great Britain, Penguim Books, 1970, 416 p.

(6) - CODD, E.F. A data Base Sub-language founded on The Relational Calculus, Proc. 1971, ACM SIGFIDET Workshop on Data Description Access and Control, San Diego, November 1971, pp. 35-68.

(7) - CODD, E.F. A Relational Model of Data for Large Shared Data Banks, Comm. ACM, vol. 13, nº 6, June 1970, pp. 337-387.

(8) - CODD, E.F. Normalised Data Base Structures: A Brief Tutorial, Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, November 1971, pp. 1-17.

(9) - DAHL, O.J., DIJKSTRA, E.W., HORRE, C.A.R. - Structured Programming, London, Academic Press, 1972, 222 p.

(10) - DIETMEYER, Donald L. - Logic Design of Digital Systems, Boston Ally and Bacon, 1971, pp. 128-131.

(11) - FERREIRA, Aurélio Buarque de Hollanda - Pequeno Dicionário Brasileiro da Língua Portuguesa, Rio de Janeiro, Civilização Brasileira, 1974, 1304 p.

(12) - GRIES David - Compiler Construction for Digital Computer, New York, John Wiley & Sons, 1971, pp. 103-106.

(13) - HABERKORN, Ernesto M. - Introdução à Análise de Sistemas, São Paulo, Atlas, 1973, 306 p.

(14) HARRISON, Michel A. - Introduction to Switching and Automata Theory, New York, McGraw-Hill, 1965, pp. 1-70.

(15) - HEATH, I.J. - Unacceptable File Operations in a Relational Data Base, Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, November 1971, pp. 19-33.

(16) - HENNIE, Frederich C. - Finite-State Models for Logical Machines, John Wiley & Sons, New York, 1968, pp.31-41.

(17) - IBM-Application Programm 1130, Commercial Subroutine Package (1130-SE-25X), Version 3, Programm Reference Manual H20-0241-3, White Plains, N.Y., IBM Corporation, November 1968, 200 p.

(18) - IBM Information Management System 1360, Application Description Manual, GH20-0524-2, White Plains, N.Y., IBM Corporation, May 1970, 42 p.

(19) - IBM Sistema 1360-Fundamentos de Programação COBOL-SR17--0205-1, Rio de Janeiro, IBM do Brasil, Setembro de 1972 pp.99.

(20) - INTEL - MSc.-8 Users Manual, Santa Clara, California, Intel Corporation,April 1975, 56 p.

(21) - KNUTH, Donald E. - Fundamental Algorithms, Vol I, Menlo Park, California, Addison-Wesley, 1969, pp. 1-227.

(22) - KNUTH, Donald E. - Fundamental Algorithms, Vol III, Menlo Park, California, Addison Wesley, 1970, pp.181-195.

(23) - KNUTH, Donald E. - Structured Programming with GO TO Statements, Computing Surveys, Vol VI, nº 4 December 1974, pp. 261-301.

(24) - MIMSKY, Marvin L. - Computation:Finite and Infinite Machines, Englewood, New Jersey, Prentice-Hall, 1967, pp.67-99.

(25) - N. C. E. / UFRJ - Sistema Operacional de Simulação-Manual do Usuário, Rio de Janeiro, Núcleo de Computação Eletrônica/Universidade Federal do Rio de Janeiro, 1975.

(26) - RANDELL, B.; RUSSELL, L.J. - Discussions on Algol Translation and Mathematic Centrum English Eletric Report W/AT 841, 1962 .

(27) - SILVA, Eduardo Dória - Relatório para COPERBO, Cabo Pernambuco, COPERBO, 1973, p.p 11-12.

(28) - WAGNER, Peter - Introduction to System Programming, N.Y. Academic Press, 1964, pp. 122-136.

(29) - WIRTH, Niklaus - Systematic Programming: An Introduction Englewood, New Jersey, Prentice-Hall, 1973, 170 p.