UM SISTEMA BASIC MULTIUSUÁRIO PARA O COBRA-400

MARIA TERESA AZEVEDO CÉTOLO

ORIENTADOR

PROF.DR. CÉLIO CARDOSO GUIMARÃES

Dissertação apresentada no Instituto de Matemática,Estatística e Ciência da Computação, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Este trabalho teve auxílio financeiro e material da FAPESP , CNPq, FINEP, COBRA-Computadores Brasileiros S/A e IBM do Brasil

Novembro - 1980

AGRADECIMENTOS

Ao Prof. Dr. Célio Cardoso Guimarães, por sua atenção, disponibilidade e segura orientação na elaboração desse trabalho.

Ao colega e amigo Daniel Andrade pelos ensinamentos técnicos na implementação do sistema BASIC.

A todos aqueles que direta ou indiretamente me apoiaram e incentivaram.

Aos meus pais Newton e Vitalina

que me deram raízes, asas

e amor.

RESUMO

Neste trabalho é descrita a implementação de um sistema multiusuário ("Time sharing") em torno da linguagem BASIC escrito em linguagem de montagem do Intel 8080. São elaboradas as modificações no interpretador para torná-lo reentrante, as estruturas de dados para compartilhamento de memória, o sistema de escalonamento do processador e as rotinas de Entrada/Saída para interface com o "hardware" e com o sistema de arquivos do COBRA - 400.

ABSTRACTS

This paper describes an implementation of a BASIC time sharing system on a COBRA-400. The system was developed starting from a BASIC Interpreter written in Intel 8080 Assembly / language.

We describe the changes in the interpreter in order to make it reentrant, the data structures for sharing memory among user processes and the scheduler organization. Input/Output / routines were also rewritten for compatibility with the COBRA-400 hardware and file system.

SUMÁRIO

INTRODUÇÃO

"Software" de aplicação tem sido desenvolvido em inúmeras linguagens. Entretanto, companhias comerciais quando desenvolvem projetos que devam ser instalados em máquinas diversas , procuram desenvolvê-los em uma linguagem que seja largamente utilizada. Podemos classificar "software" de aplicação em duas áreas : científica e comercial. Na primeira FORTRAN e BASIC são linguagens de aceitação geral. Na segunda COBOL e RPG.

É muito provável que a massificação do uso de microprocessadores não cause mudanças radicais nesse padrão de uso, e que "software" de aplicação a ser desenvolvido nas novas máquinas se volte para as linguagens existentes.

Na área de aplicações científicas a linguagem BASIC ("Beginner's All-Purpose Symbolic Instruction Code") [1],desenvolvida em meados de 1960 no "Dartmouth College", sob a direção de John Kemeny e Thomas Kurtz, inicialmente para fins educacionais, tornou-se extremamente difundida, devido a sua simplicidade, facilidade de implementação e caráter interativo. Isto se deve ao fato de que a maioria das implementações não gera código objeto, mas interpreta diretamente o programa fonte, possibilitando combinar, de maneira simples e compacta, as facilidades de edição de programa / com as de cálculo, de forma interativa e natural, à custa de menor

velocidade de execução. Em muitas aplicações científicas de enge - nharia, de controle de processos e aplicações comerciais, esta limitação é aceitável.

Este trabalho descreve a adaptação do Inter - pretador BASIC, desenvolvido pelo Laboratório de Lawrence Livermore [2], às peculiaridades do minicomputador COBRA-400 [3], e a implementação de um sistema de tempo repartido em torno do interpretador para uso com quatro terminais (estendível, facilmente, para 8 terminais) de forma interativa e concomitante.

O espaço de memória requerido é de 6 "Kbytes" para o interpretador e rotinas de ponto-flutuante, 3 "Kbytes" para rotinas de entrada e saída e "buffers", 2 "Kbytes" para variáveis / de controle e pilha dos usuários num total de 11 "Kbytes". O espaço disponível para armazenar programas fonte é de 50 "Kbytes", o que é suficiente, por exemplo, para um usuário inserir e executar / um programa de 1500 linhas de 25 caracteres cada uma, 10 vetores de dimensão 100 e 200 variáveis simples.

No capítulo 1 damos uma breve descrição do subconjunto implementado da linguagem BASIC, o funcionamento do interpretador e algumas características do COBRA-400, componentes / básicos para a implementação do sistema BASIC monoprogramado.

Os capítulos subseqüentes se referem à implementação do sistema Multiusuário . O capítulo 2 discute opções para introdução de reentrância no interpretador original e a eficiên-

cia do sistema em termos de tempo de execução e memória utilizada . O capítulo 3 apresenta o sistema de gerenciamento e proteção de memória. O capítulo 4 trata do sistema de interrupção, entrada e saída tanto do teclado como do disco fixo. O capítulo 5 mostra o escalonamento de processos, isto é, a criação de um sistema de execução para administração do tempo de processamento.

Finalmente, no capítulo 6, são apresentadas/ medidas de desempenho do sistema Multiprogramado em relação ao sistema monoprogramado e são indicadas sugestões para futuras extensões.

CAPÍTULO 1

IMPLEMENTAÇÃO DO INTERPRETADOR BASIC MONOPROGRAMADO NO COBRA-400 .

1.1 - INTRODUÇÃO

O Interpretador BASIC foi projetado para executar no microcomputador MCS-8080 originalmente, e está escrito em linguagem de montagem do Intel-8080. Devido ao seu pequeno tamanho (5 "Kbytes") a implementação do Laboratório de Lawrence Livermore / foi de um subconjunto bastante simplificado do BASIC, que foi chamado de BASIC-8080 . Uma característica particular dessa implementação foi a inclusão de chamadas especiais para rotinas do usuário escritas em linguagem de montagem.

1.2 - A LINGUAGEM BASIC-8080 [4] .

Denominaremos ações as diretivas do usuário / para o interpretador. Três ações foram implementadas originalmente:

RUN - Execute o programa corrente armazenado na memória.

SCR - Apague o programa corrente.

LIST - Mostre o programa corrente no vídeo.

O interpretador original possui as ações PLST (perfura cópia do programa corrente em fita de papel) e PTAPE (lê

programa da fita de papel). Como o COBRA-400 não dispõe de leitora/perfuradora de fita de papel e também por razões de eficiência essas duas ações foram substituídas , respectivamente, pelas ações/ WRITE e READ sobre o disco fixo.

Chamaremos de comandos as linhas fontes do programa usuário. Todos os comandos da linguagem BASIC-8080 requerem um rótulo numérico, denominado número de linha ou número de comando. A ordem de execução dos comandos é a ordem crescente dos respectivos números de linha.

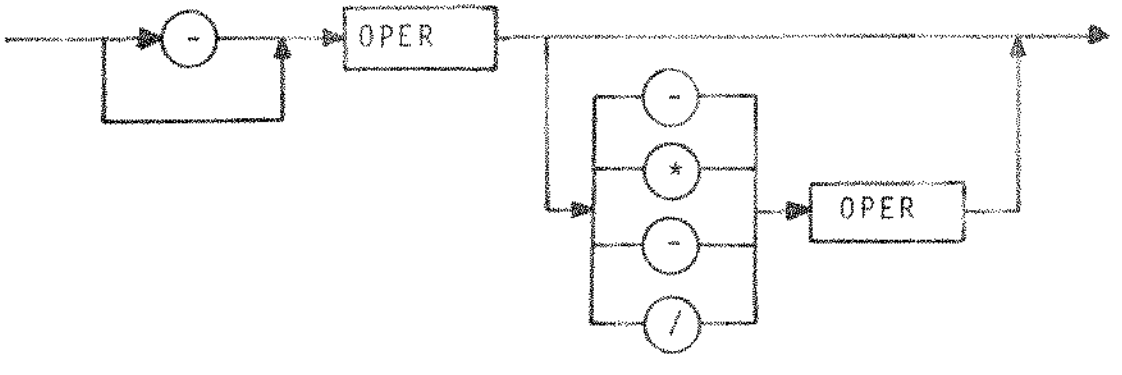
Os comandos aceitos pelo Interpretador são : REM, END, STOP, DIM, LET, IF-THEN, INPUT, PRINT, FOR-NEXT, GOSUB, RETURN, CALL, GO TO.(Ver apêndice 1)

Variáveis simples são representadas por uma letra ou uma letra seguida por um número. Variáveis dimensionadas (somente uma dimensão) devem ter apenas uma letra. É possível ter uma variável dimensionada com o mesmo nome de uma variável simples.

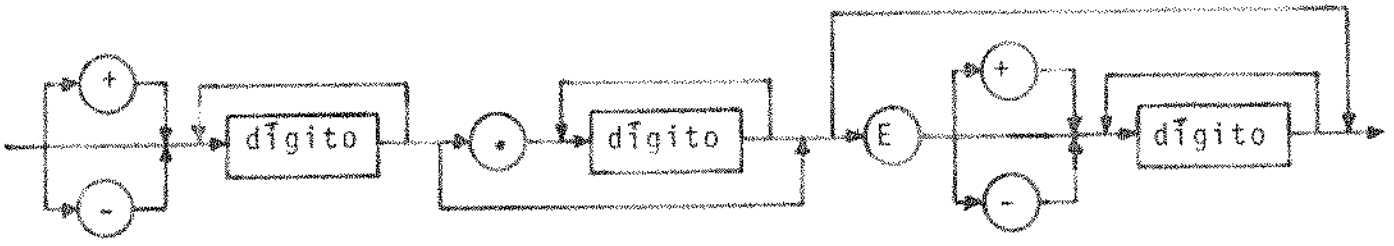
O comando de atribuição tem a forma:

número de linha LET variável = expressão aritmética

onde "expressão aritmética" contêm um ou dois operandos segundo o diagrama sintático abaixo:



onde "oper" é uma variável ou constante. As constantes, declaradas no programa BASIC ou recebidas via teclado, podem ter as seguintes formas:



O intervalo permitido para constantes é $\pm 9{,}223372 \times 10^{18}$. O menor intervalo em torno de zero é $\pm 2{,}71050 \times 10^{-20}$, porém o zero é representado exatamente.

1.3 - FUNCIONAMENTO DO INTERPRETADOR

O interpretador é constituído de duas partes: edição e execução de programas.

A primeira parte é caracterizada como um editor de texto(na verdade um editor de linha), onde o usuário insere, remove e corrige linhas do seu programa fonte.

Cada linha fonte inserida é finalizada com um caractere especial "New line". É possível corrigir erros de edição através da remoção da linha ou pela remoção do(s) caractere(s) mais

recente(s) da linha sendo criada através da tecla "RUBOUT". O comando é removido quando seu número de linha seguido pelo caractere "New line" é enviado ao interpretador.

Um comando pode ser inserido entre dois outros comandos; para isso basta criar o número de linha do comando a ser inserido com o valor entre os números de linha dos outros comandos. Para substituir um comando, o novo comando deve ter o mesmo número de linha do antigo.

A segunda parte é o interpretador propriamente dito e consiste na análise e execução dos comandos.

As ações não têm número de linha,pois indicam ordens a serem executadas e não comandos do programa do usuário.

A ação LIST pode ser seguida ou não por um ou dois números de linha indicando início e fim dos comandos a serem / transmitidos ao vídeo.

A seguinte estimativa foi dada pelos implementadores [2] para o tempo médio de execução das operações de ponto flutuante:

OPERAÇÃO	TEMPO(ms)
adição	2,4
subtração	2,4
multiplicação	5,4
divisão	7,0

1.4 - CARACTERÍSTICAS GERAIS DO COBRA-400

O minicomputador COBRA-400 disponível no laboratório de microprocessadores da UNICAMP, tem 64 "Kbytes" de memória principal, 4 terminais especiais de vídeo, unidade de disco fixo de 5MB, uma unidade de disco flexível, impressora serial e uma unidade de fita magnética.

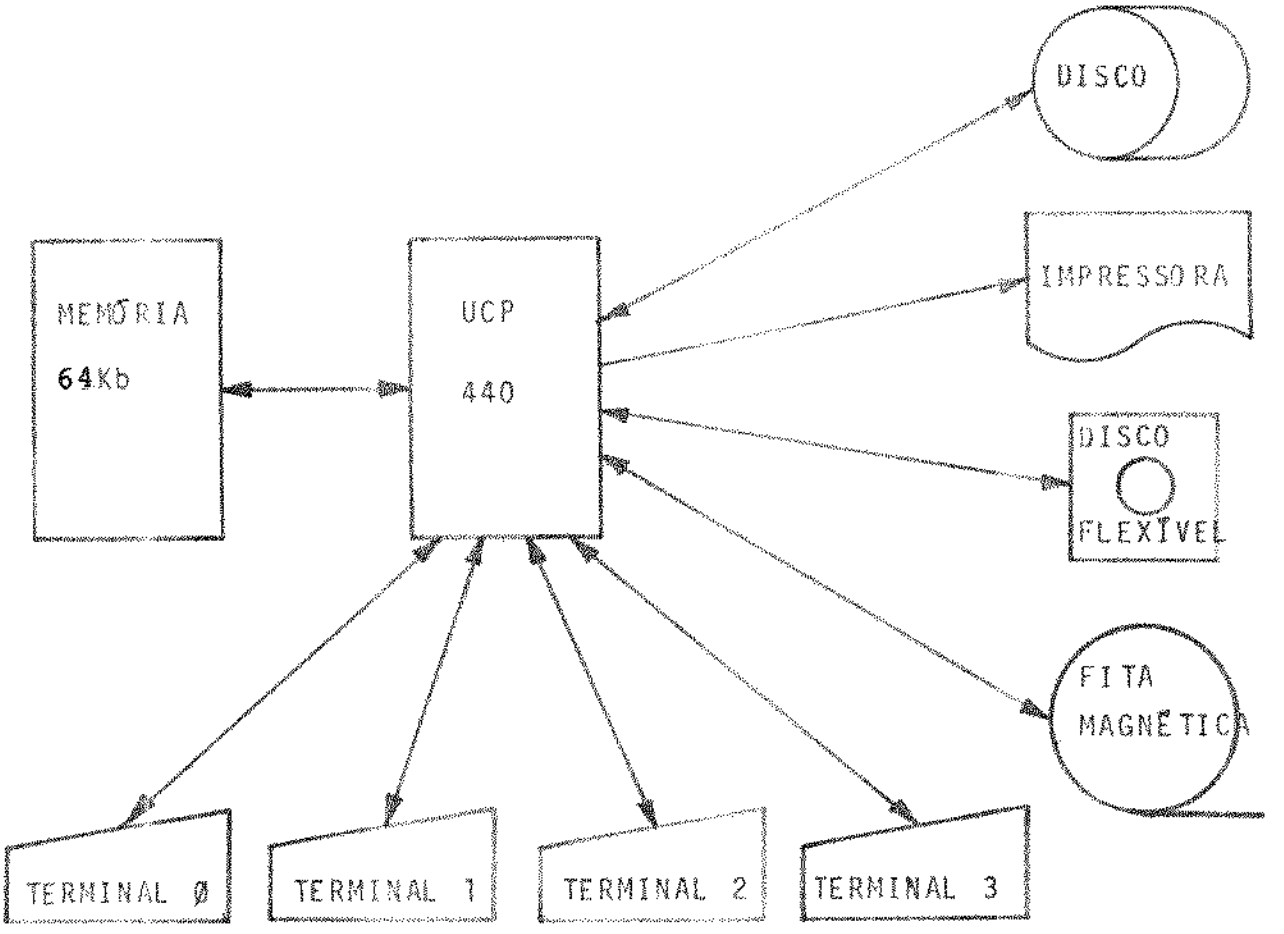


Figura 1-Sistema COBRA-400 da UNICAMP

O processador da UCP 400 é um microprocessador Intel 8080.

O teclado do terminal é o dispositivo de entrada primária do sistema. Quando uma tecla é pressionada, o módulo de controle do terminal determina o código da tecla através do circuito do teclado e gera uma interrupção. O caractere correspondente à tecla pressionada é colocado, por programa, na tela do vídeo que ocupa uma imagem na memória principal. [5]

O COBRA-400 dispõe também de um relógio de tempo real (CLOCK) não programável e que a cada 50 ms gera uma interrupção.

Para implementar o sistema BASIC monousuário, foram desenvolvidas rotinas de entrada e saída compatíveis com o COBRA-400.

CAPÍTULO 2

IMPLEMENTAÇÃO DO INTERPRETADOR REENTRANTE

2.1 - INTRODUÇÃO

Para o sistema se tornar multiprogramado a seguinte restrição foi imposta: o programa deve ser reentrante, isto é, não alterar seu próprio código(código puro) e instruções que lidam com endereços de dados devem ser substituídas de tal forma que para cada usuário as instruções se refiram ao seu conjunto de dados. Outra opção é cada usuário ter uma cópia do sistema BASIC para o seu atendimento.

A primeira opção foi escolhida por requerer um aumento de apenas 10% no tamanho do interpretador e um decréscimo / de eficiência do tempo de execução da ordem de 12%. Desta forma uma única cópia do interpretador é compartilhada por todos os usuá - rios.

O conjunto de dados do interpretador BASIC / consiste de uma "página" (256 "bytes")para as variáveis de controle e uma"página"para a pilha do usuário para operações PUSH e POP e chamadas de rotinas.

As variáveis de controle consistem basicamente de áreas temporárias do interpretador e apontadores para dados / do usuário. Uma descrição completa é dada no apêndice 2.

2.2 - RELOCAÇÃO

Denominaremos de instruções especiais as instruções que lidam com o conjunto de dados do usuário [6]:

1 - LDA adr - carregue o acumulador com o conteúdo de adr

2 - LHLD adr - carregue o par de registradores (H,L) com o conteúdo de adr e adr+1.

3 - LXI rp, adr - carregue o par de registradores indicado por rp ((H,L),(B,C), (D,E) ou (SP)) com o valor adr(16 "bits"); utilizada para carregar o endereço de uma variável no par rp.

4 - STA adr - armazene o conteúdo do acumulador (A) na posição adr.

5 - SHLD adr - armazene o conteúdo do par de registradores (H,L) nas posições adr e adr + 1.

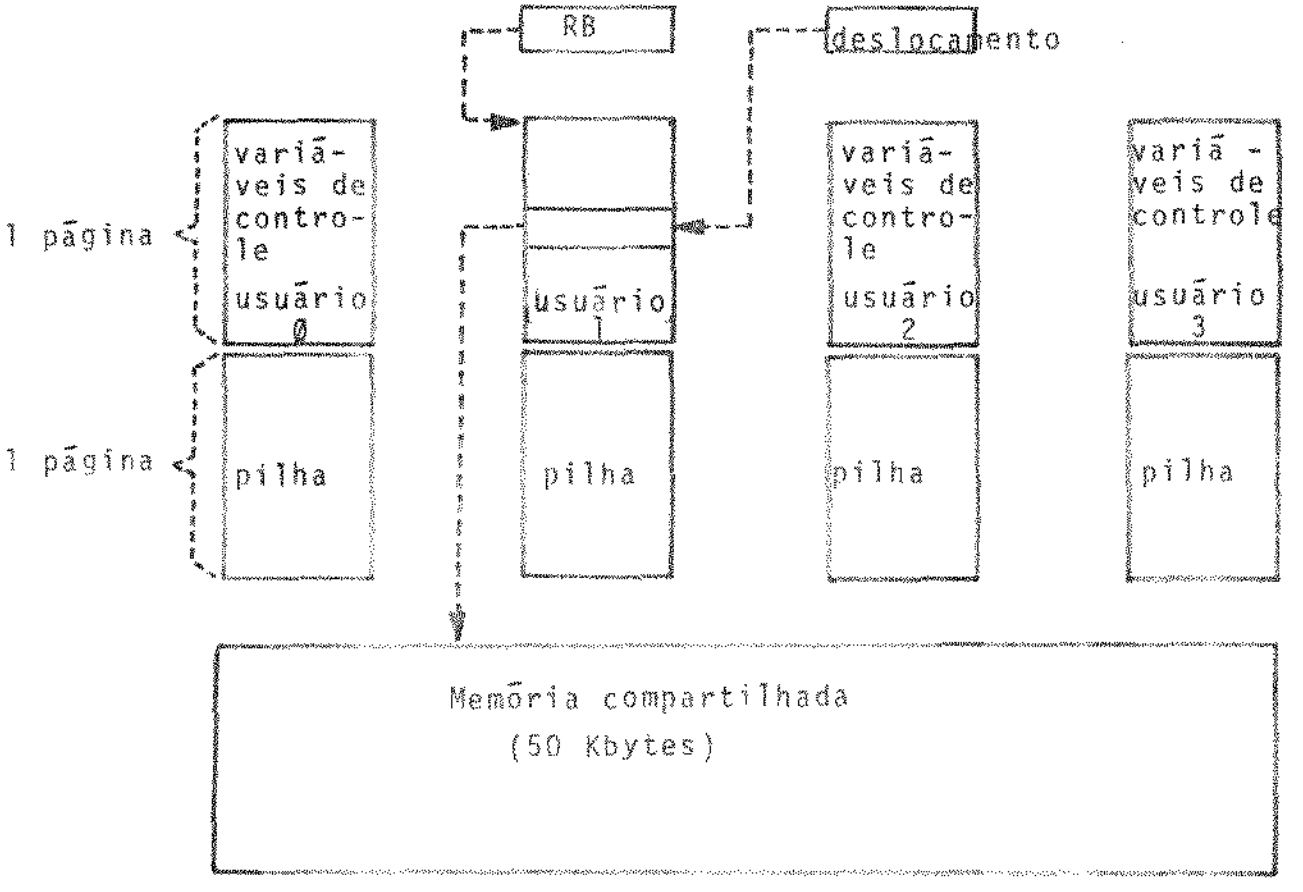
Um endereço no 8080 ocupa 16"bits" onde os 8 "bits" mais significativos representam um"número de página" e os 8 "bits" restantes indicam um deslocamento relativo à página.

Por razões de eficiência de endereçamento no 8080, no sistema monoprogramado o acesso a uma variável de controle é determinado apenas por um parâmetro: o deslocamento da variável relativo ao início de uma página pré-definida.

No sistema multiusuário, para se ter acesso

a uma variável de controle, o interpretador terá dois parâmetros : número da página, deslocamento. Com cada usuário foi associado um número de página que identifica univocamente as variáveis de controle do usuário. Em tempo de execução do sistema BASIC, o número de página estará contido numa posição de memória que chamaremos de registrador base (RB) em analogia com a sua função em máquinas que possuem relocação dinâmica.

O deslocamento define uma variável de controle dentro do conjunto; assim uma determinada variável de controle/ possui o mesmo deslocamento para todos os usuários.



As instruções especiais foram substituídas por um conjunto de instruções que manipulam o par - número de página,deslocamento - que define uma variável de controle de um usuário.

As modificações visaram dois objetivos: economia de memória e economia de tempo de execução. Duas opções foram / consideradas: através de macros (substituição direta), que consiste em substituir cada ocorrência da instrução por um conjunto de instruções e substituir cada ocorrência da instrução por uma chamada de rotina.

Em seguida apresentamos uma análise das duas opções para a instrução LDA (a análise para todas as instruções especiais aparece no apêndice 4):

- Instrução LDA adr

Ocorrência estática no interpretador : 8 vezes

A. Substituição Direta

		Nº de "Bytes"	Nº de Ciclos
PUSH	H	1	11
LHLD	RB	3	16
MVI	L,adr	2	10
MOV	A,M	1	7
POP	H	1	10
		8	54
	Nº TOTAL	64	432

Observe que, se o endereçamento utilizado não fosse relativo à página, as instruções para substituição direta seriam , por exemplo:

INSTRUÇÕES		Nº de "Bytes"	Nº de Ciclos
PUSH	D	1	11
PUSH	H	1	11
LHLD	RB	3	16
LXI	D,adr	3	10
DAD	D	1	10
MOV	A,M	1	7
POP	H	1	10
POP	D	1	10
		12	85

O endereçamento relativo à página foi possível graças ao pequeno número de variáveis de controle, ficando todas agrupadas em uma mesma página para cada usuário.

B. Através de Chamada de Rotina

	MVI	A,adr	2	7
	CALL	LDAA	3	17
	⋮		5	
LDAA:	PUSH	H	1	11
	LHLD	RB	3	16
	MOV	L,A	1	5

MOV	A,M	1	7
POP	H	1	10
RET		1	10
		8	83
	Nº TOTAL	48	664

Um resumo do número total de "bytes" é apresentado na tabela a seguir:

Instrução	Nº de "Bytes" Substituição Direta	Nº de "Bytes" Chamada Rotina	"Bytes" utilizados pelo Interpretador Original
LDA	64	48	24
STA	32	36	12
LHLD	671	378	183
SHLD	442	283	102
LXI	323	323	174
	A = 1532	B = 1068	C = 495

O aumento real do número de "bytes" requerido é, portanto :

1) Substituição Direta

$A - C = 1532 - 495 = 1037 \cong 1$ "Kbyte"

2) Substituição Através de Rotinas

$B - C = 1068 - 495 = 573 \cong 1/2$ "Kbyte"

O tempo de execução do programa pode ser medido pelo número de ciclos de cada instrução multiplicado pelo número de vezes em que ela é executada.Este tipo de anâlise dinâmica é complexo dado o número de entradas que o programa pode admitir. A análise estática do número de ciclos apresentada abaixo é, portanto, uma aproximação grosseira do aumento do tempo de execução numa aplicação real:

INSTRUÇÃO	Nº de Ciclos Substituição Direta	Nº de Ciclos Chamada de Rotina	Nº de Ciclos no Interpretador Original
LDA	432	664	104
STA	216	428	52
LHLD	4270	6466	976
SHLD	3094	4182	544
LXI	1783	1783	580
	A = 9795	B = 13523	C = 2256

Aumento do número de ciclos:

1) Substituição Direta

A - C = 9795 - 2256 = 7539

2) Substituição Através de Rotinas

B - C = 13523 - 2256 = 11267

O sistema BASIC tem cerca de 4000 instruções. Calculando-se a porcentagem do aumento do número de ciclos em fun -

ção do número total de instruções do sistema, usando uma média de 10 ciclos por instrução, temos um aumento estático de aproximadamente 19% no caso de susbtituição direta, e 28% no caso de substituição por rotinas.

A diferença entre o aumento no número estático de ciclos da opção de substituição direta e por chamada de rotina é, pois, da ordem de 9%.

A substituição das instruções por chamadas de rotinas dá maior economia em termos de memória e maiores facilidades de depuração. Pelas razões acima, foi escolhido o método de substituição através de chamadas de rotinas.

Na prática, obteve-se um aumento do tempo de execução de programas típicos em relação ao sistema original mono-programado de 10% a 15% e o resultado dessas medidas experimentais é apresentado no capítulo 6.

CAPÍTULO 3

GERENCIAMENTO DE MEMÓRIA

3.1 - INTRODUÇÃO

Com a introdução do sistema multiusuário a memória será compartilhada por todos os usuários. Uma solução para compartilhar a memória seria dividi-la em quatro partes iguais, uma para cada usuário. Neste caso, surge uma séria limitação do sistema: se um usuário fizesse um programa, cujo tamanho fosse maior que sua partição , seria limitado por falta de espaço,quando, na realidade, poderia usar parte da memória das outras partições que no momento estivesse disponível.

Uma solução mais flexível é fazer alocação / dinâmica de memória de tal forma que os usuários pudessem ter programas de tamanhos variados, limitados é claro, pelo espaço total de memória existente.

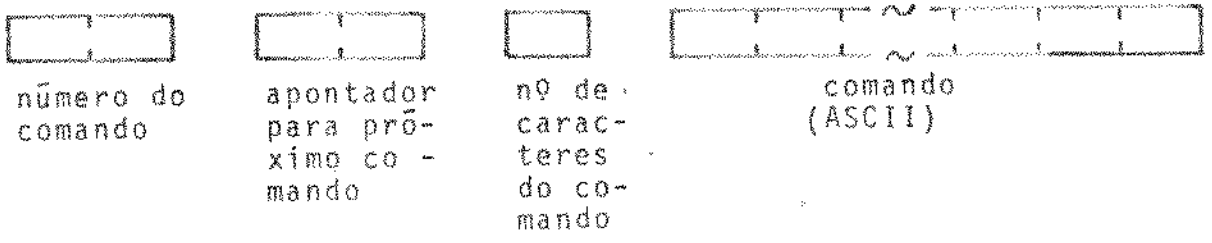
A administração de memória adotada permite a cada usuário alocar, sob demanda, espaço de uma área global de memória. A organização do programa fonte sob a forma de lista ligada torna bastante simples o gerenciamento dessa área global.

3.2 - MODIFICAÇÕES NA ESTRUTURA DO INTERPRETADOR

No sistema BASIC, as linhas do programa do

usuário, ao serem criadas, são compactadas pelo interpretador(caracteres brancos são eliminados, exceto os incluídos entre aspas) e armazenadas na memória sob a forma de uma lista ligada.

Cada componente desta lista é um registro com a seguinte forma (cada retângulo corresponde a um "byte"):



A rotina do interpretador (TTYIN) que armazena o programa fonte na memória faz:

1 - Aloca 5 posições para as informações do registro;

2 - Lê caractere (detectado pela rotina de interrupção);

3 - Se caractere é "ERROR RESET " então

- remove a linha lida ;

- retorna

4 - Se caractere é "RUBOUT" então

- remove o último caractere armazenado ;

- vá para 2

5 - Se caractere é "NEW LINE" então

- faz a compactação dos caracteres;
- retorna

Senão

- armazena caractere na memória;
- vá para 2

Um problema com o algoritmo acima é que interrupções provenientes de outros usuários podem requerer também alocação de memória e não haverá garantia de que os caracteres de entrada sejam armazenados seqüencialmente dentro do registro.

Inibir interrupções no início da rotina e armazenar todos os caracteres que formam a linha do comando e ativar interrupções antes do retorno não é uma solução razoável, pois o sistema ficaria bloqueado para os outros usuários e o processador ocioso à espera de caracteres desse usuário.

A solução adotada foi alterar a rotina (TTYIN) para que os caracteres de uma linha sejam armazenados num "buffer" local para cada usuário. Ao encerrar a edição desta linha, é feita a transferência dos caracteres do "buffer" auxiliar para a lista do usuário na área de memória global. Por razões óbvias esta operação de transferência deve ser executada como uma região crítica [7].

A exclusão mútua é garantida de maneira simples, num sistema com um só processador, através de inibição de in

terrupções no começo da região crítica e ativação de interrupções no final da região crítica.

As posições livres estão alocadas em uma região contígua de memória, sendo que a primeira posição desta região é apontada por uma variável de controle global(NLINE) do interpretador. Se um determinado usuário requer n posições de memória, para inserir uma linha de seu programa, são alocadas n posições a partir da variável NLINE cujo valor é incrementado de n.

O algoritmo da rotina TTYIN foi modificado para:

1 - Lê caractere (detectado pela rotina de interrupção);
2 - Se caractere é "ERROR RESET" então
    - remove a linha que está sendo inserida;
    - retorna
3 - Se caractere é "RUBOUT" então
    - remove o último caractere lido;
    - vá para 1
4 - Se caractere é "NEW LINE" então
    - inibe interrupções ; (começo da região crítica)
    - aloca 5 posições para informações do registro;
    - faz a compactação da linha lida;
    - transfere linha do "buffer" auxiliar para memória ;
    - atualiza NLINE para próxima posição após registro ;
    - ativa interrupções ; (fim da região crítica)
    - retorna

Senão

- armazena caractere no "buffer" auxiliar;
- vá para 1.

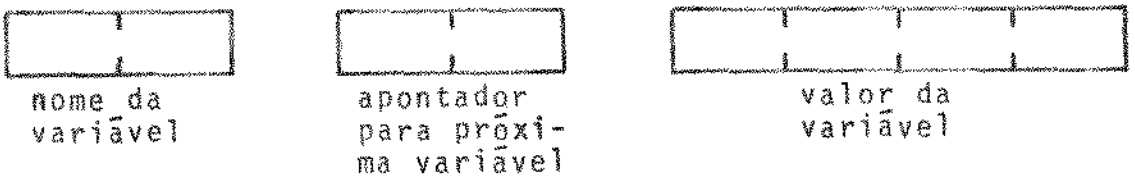
Antes de transferir a linha do "buffer" auxiliar para a memória é feita uma análise para verificar se se trata de uma ação ou de um comando. Esta análise é feita identificando-se o primeiro caractere da linha, que, se for um número então a linha é um comando, ou, se for letra, a linha representa uma ação. No caso de ser uma ação será interpretada imediatamente sem fazer a transferência para a memória. Note que no caso de comando não é feita análise sintática.

Outro problema é compartilhar a memória com as tabelas de símbolos dos usuários.

A tabela de símbolos é construída em tempo / de execução e tem início após o programas fonte do usuário.

As entradas da tabela são as seguintes:

VARIÁVEL SIMPLES



VARIÁVEL DIMENSIONADA

nome do vetor

apontador para próxima variável

elementos do vetor

Uma variável do programa fonte do usuário é inserida na tabela de símbolos quando é referenciada pela primeira vez durante a interpretação de um comando, exceto variáveis dimensionadas que devem ser declaradas previamente ao seu uso através / do comando DIM e são inseridas nesta ocasião.

Busca e inserção na tabela de símbolos do usuário são feitas pelo algoritmo seguinte:

1 - Se tabela de símbolos vazia então

1.1 - se variável dimensionada então retorna

- inibe interrupções;(começo da região crítica)
- a partir da posição de memória apontada por NLINE ,insere nome da variável, apontador para próxima variável e reserva 4 posições para o valor;
- atualiza apontadores da lista;
- atualiza NLINE (NLINE ← NLINE + 8);
- ativa interrupções; (fim da região crítica)
- retorna com endereço do valor da variável

Senão

- busca variável na tabela de símbolos;
- se variável está na tabela de símbolos então
  - retorna com endereço do valor da variável

Senão vá para 1.1

A rotina que analisa o comando DIM foi alterada para operar de maneira análoga a de uma variável simples:

1 - Inibe interrupções;

2 - A partir da posição de memória apontada por NLINE insere nome da variável, apontador para a próxima variável;

3 - N ← dimensão da variável;

4 - Se N = 0 então
  - ativa interrupções;
  - retorna

Senão
  - reserva 4 posições de memória para valor do elemento;
  - NLINE ← NLINE + 4 ;
  - N ← N - 1 ;
  - vá para 4

Observe que todos os componentes de um vetor ocupam posições consecutivas de memória.

3.3 - PROTEÇÃO DE MEMÓRIA

Com a introdução do sistema de compartilhamento de memória, o problema de interferência destrutiva é evidente. Um erro de um programa do usuário poderia facilmente destruir programas dos outros usuários. A proteção de memória se fez necessária, garantindo ao usuário que interferências alheias de qualquer natureza não afetam o curso de seu programa.

Foram implementados três tipos de proteção: contra acesso ilegal de memória, memória com capacidade esgotada e pilha do usuário esgotada.

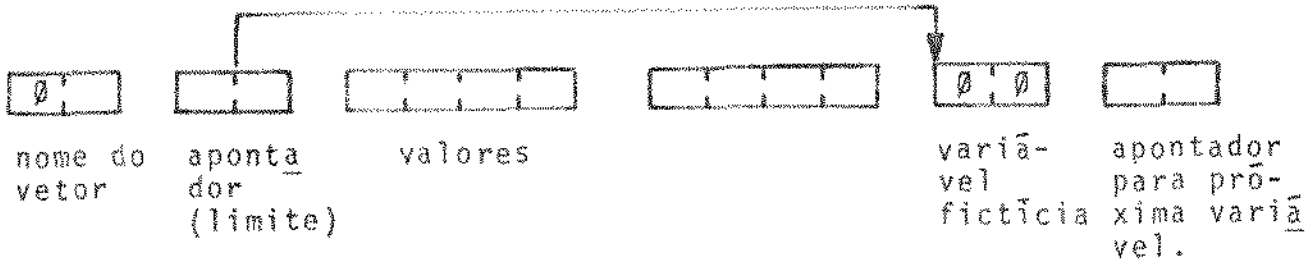
O acesso ilegal à memória só pode ser feito/ através de uma atribuição a uma variável dimensionada, cujo índice estaria fora dos limites permitidos.

No sistema monoprogramado este tipo de atribuição é realizada e erros podem ocorrer.

Para solucionar este tipo de erro foi necessário mudar a estrutura das variáveis dimensionadas, de tal forma/ que o limite do índice fosse incluído na nova estrutura. Por outro lado, não se queria alterar o esquema existente de tratamento de variáveis dimensionadas. Para atender às duas exigências não / se alterou a estrutura pré-definida e na declaração de uma variável dimensionada foi introduzida, imediatamente após a alocação de espaço para todos os seus elementos, uma"variável" fictícia , cuja função é delimitar o final dos elementos do vetor. Dessa for

ma, o "apontador para a próxima variável" do vetor declarado aponta para esta variável fictícia.

A nova estrutura de variáveis dimensionadas é então:



A verificação do índice I de uma variável dimensionada X é feita da seguinte forma:

1 - Busca na tabela de símbolos o 1º elemento do vetor X ;

Ender ← endereço do 1º elemento (X (0))

2 - Cálculo do endereço de X(I);

EL ← Ender + 4 * I

3 - Cálculo do limite do vetor;

LIM ← (Ender - 2) - 4

4 - Verificação se índice dentro do limite permitido

Se LIM < EL então

- Envia mensagem:"Índice do vetor fora do limite"
- Pára execução

Senão retorna OK

Quando uma linha de um programa fonte é removida pelo usuário, o que ocorre na realidade é que o apontador para esta linha é eliminado da lista ligada e o espaço da linha não é recuperado. A ação "SCR" - apague programa da memória - apenas/ elimina os apontadores para este programa que continua residente na memória. O mesmo ocorre com as tabelas de símbolos. Após a execução do programa, a tabela é desativada, ou seja, seus apontadores serão destruídos embora a tabela continue na memória. (A cada execução será ativada uma nova tabela).

Uma maneira de se recuperar estas posições / livres de memória é colocá-las sob a forma de lista ligada, onde cada componente desta lista é um registro contendo: apontador para próximo registro, número de posições livres e, em seguida, as posições livres do registro. Esta solução traz dois inconvenientes: primeiro, a rotina que faz a alocação de posições de memória a cada pedido de alocação percorre a lista ligada para obter o registro adequado para o número de posições desejadas, e para liberar as posições de memória é preciso atualizar esta lista ligada; segundo, como os registros são de tamanhos variados é possível que na alocação de memória nenhum registro tenha o número de posições requeridas.

Outra solução é fazer uma compactação dos / programas dos usuários quando a variável de controle NLINE apontar fim de memória. Porém a complexidade deste processo não é com

pensada dado que o espaço de memória recuperado é , em geral, pe - queno causando ,nas próximas alocações ,este tipo de erro novamente.

A solução adotada quando ocorre o erro de me mória esgotada é gravar o programa fonte dos usuários no disco fixo e o usuário reinicializa o sistema BASIC colocando novamente os programas na memória,através da ação READ , de forma a reapro - veitar os espaços inutilizados. A reinicialização do sistema pelo usuário foi adotada porque outros usuários podem ter seus progra - mas em execução sem precisar alocar mais memória. Dessa forma / èles não seriam afetados pela falta de memória,mas apenas o usuá - rio corrente. É claro, ele deverá, nesse caso, pedir autorização aos outros usuários para reinicializar o sistema. Uma reinicializa- zação automática iria requerer que o sistema esperasse que todos os usuários entrassem num estado especial "à espera de ação, ou entra da de comando".

Um usuário pode esgotar o limite da sua pi - lha através de chamadas aninhadas de rotinas. Na interpretação / dos comandos GOSUB e CALL, é testado o valor limite da pilha dado pela constante global STKFULL que representa um deslocamento cons- tante em relação ao fim da área reservada para a pilha.

A proteção de memória foi implementada ape - nas contra erros de programas totalmente escritos em BASIC, sem cha madas a rotinas escritas em linguagem de montagem através do coman do especial CALL já que tais rotinas são simplesmente executadas e não interpretadas.

CAPÍTULO 4

ENTRADA E SAÍDA MULTIPROGRAMADA

4.1 - INTRODUÇÃO

O contexto do processo de um usuário é o conjunto de informações privativas do processo necessárias para sua execução. Este contexto global é constituído das seguintes partes:

1 - Variáveis de controle

2 - Programa fonte

3 - Tabela de símbolos

4 - Pilha e apontador de pilha

5 - Registradores, códigos de condição, contador de instruções

6 - Tabela de entradas para as sub-rotinas em linguagem de montagem (usada pelo comando CALL)

7 - Arquivos privativos

A preservação do contexto do usuário é fundamental para o funcionamento correto do sistema multiprogramado, especialmente durante o serviço de interrupções que necessariamente / destrói a parte do contexto que reside no processador e que denominaremos de "contexto imediato" do processo:

- Registradores (H, L, B, C, D, E, A)

- Códigos de condição (PSW)

- Apontador de pilha (SP)
- Contador de instruções (PC)

4.2 - INTERRUPÇÃO DO TECLADO

Interrupções no sistema BASIC multiusuário / provêm dos teclados e do relógio de tempo real.

Quando o pedido de interrupção é aceito, o processador inibe interrupções, desvia para a posição de memória 8 salvando o contador de instruções na posição indicada pelo registrador SP ("STACK POINTER"). Esse procedimento é chamado "mecanismo de interrupção".

Na posição 8 de memória tem-se um desvio para a rotina de interrupção propriamente dita (Ver também apêndice 5):

1 - Salva o contexto imediato do programa interrompido numa pilha da rotina de interrupção.

2 - Determina, através da leitura do estado do "port" 0 , se a interrupção foi do teclado ou do relógio:

A - Interrupção do Teclado:

3 - Determina qual dos teclados interrompeu através de leitura do "port" 3EH.

4 - Desliga o pedido de interrupção do teclado através da saída no "port" 3EH.

5 - Lê o código da tecla pressionada através de leitura do "port" 3FH

6 - Converte o código da tecla para o código ASCII

7 - Insere o caractere no "buffer" circular correspondente ao teclado

8 - Ecoa o caractere no vídeo correspondente

9 - Chama o escalador de tarefas. (Capítulo 5)

B - Interrupção do relógio - Veja Capítulo 5,parágrafo 5.3

O código de uma tecla é composto de 1 "byte" onde os 2 "bits" mais significativos representam o código da tecla auxiliar (caso tenha sido pressionada conjuntamente) e os 6 "bits" restantes representam a posição da tecla no teclado.

TECLA AUXILIAR	CÓDIGO
SHIFT	00
FUNCTION SELECTION	01
I/O CONTROL	10
nenhuma	11

Através da rotina de interrupção do teclado , os usuários do sistema BASIC fazem a inserção dos caracteres que serão transferidos para o interpretador.

4.3 - ENTRADA E SAÍDA VIA TERMINAL

A rotina"Entrada" transfere os caracteres / do "buffer" do terminal para o interpretador. O parâmetro exigido por esta rotina é o número do usuário que pede o dado. A cada ter

minal (usuário) é associado um número(0, 1, 2 ou 3). Usando o número do usuário como índice numa " Tabela de contexto" obtêm-se os apontadores "começo" (CB) e "fim" (FB) do "buffer" circular do usuário:

Se CB = FB então

chama escalador de tarefas, pois o "buffer" está vazio e o processo do usuário está à espera de caracteres

Senão

CB ← CB + 1 mod 64

(A) ← BUFF [CB]

Observe que quando o buffer está vazio, ao invés de entrar num laço à espera de dados, como acontecia no sistema monoprogramado, a rotina de "Entrada" salva o contexto imediato e chama o escalador de tarefas. Se a transferência para outro processo (usuário) não ocorre, então este processo entra num laço à espera de caracteres. (Veja apêndice 6).

A transferência de dados do interpretador para o usuário, como mensagens de erros, listagem de programas e resultados, é feita através da rotina SAÍDA. Os parâmetros de entrada desta rotina são: número do usuário e caractere a ser transmitido.

O vídeo de um terminal é uma imagem da memória e ocupa 576"bytes"(240H). A imagem do terminal 0 (zero) começa

na posição de memória 700H e vai até 93FH. A imagem do terminal 1 começa na posição 940H e vai até B7FH , e assim por diante.

O vídeo é composto de nove linhas de 64 caracteres cada uma. Qualquer caractere será enviado para a linha mais inferior do vídeo na posição corrente do cursor que avança uma posição para a direita com exceção do caractere especial "RUBOUT" que causa o retorno do cursor de uma posição e do caractere "NEW LINE" que causará um deslocamento de todas as linhas para cima (a 1a linha desaparece) deixando a nona linha em branco para receber caracteres.

Um caractere enviado quando o cursor está na última posição da linha tem o mesmo efeito que o caractere " NEW LINE".

O parâmetro "número do usuário" identifica o vídeo associado ao usuário.

Note que tanto os "buffers" circulares como os vídeos dos terminais são usados por processos concorrentes. Neste caso, o acessó às variáveis, como cursor e apontadores do "buffer" é feito dentro de uma região crítica cuja implementação é análoga à do sistema de gerenciamento de memória.

4.4 - CARACTERES DE CONTROLE DO VÍDEO

Os caracteres especiais que permitem con -

trolar a quantidade de linhas transmitidas para o vídeo são:

"CONTROL Y" - (Tecla "I/O CONTROL" + Tecla Y) - corta transmissão de dados.

"CONTROL S" - (Tecla "I/O CONTROL" + Tecla S) - inibe transmissão de dados.

"CONTROL Q" - (Tecla "I/O CONTROL" + Tecla Q) - ativa a transmissão de dados.

(A função de "CONTROL S" e "CONTROL Q" é idêntica à do sistema / TOPS1Ø e a do "CONTROL Y" é análoga à do "CONTROL C").

Esses caracteres são detectados pela rotina de interrupção e seu valor é colocado numa variável de controle do interpretador. Não são transmitidos através de rotina "Entrada " porque, como são caracteres de controle,devem ter prioridade na transmissão para o interpretador.

Após cada linha transmitida ao vídeo a rotina de controle QUITT é chamada e se existe um "CONTROL Y" então / pára a execução do processo do usuário. Se "CONTROL S" então chama o escalador de tarefas para transferir o controle para outro usuário, ficando este processo à espera de um "CONTROL Q".

A verificação de caracteres de controle só é feita durante a execução das ações "LIST" e "RUN".

Apesar da existência dos caracteres de controle de vídeo não é possível ao usuário analisar a listagem do

seu programa fonte devido à grande velocidade com que as linhas / são transmitidas para o vídeo pela ação LIST. Por isso, esta ação foi implementada de tal forma que são transmitidas oito li - nhas de cada vez. Se o usuário quiser continuar a transmissão de ve enviar "CONTROL Q" e o processo da ação LIST se repetirá para as próximas oito linhas e assim, sucessivamente, até o final do programa.

4.5 - ENTRADA E SAÍDA VIA DISCO FIXO

4.5.1 - CONTROLE DO DISCO FIXO

Além do microprocessador 8080 da UCP há um outro 8080 no controlador de disco, com memória auxiliar indepen dente da memória principal [8]. A comunicação entre a "UCP" e o controlador do disco fixo é feita através dos primeiros 32K da memória principal que correspondem aos últimos 32K de endereçamen to da memória auxiliar do controlador, havendo, portanto, uma so- breposição parcial dos espaços de endereçamento.

Nesta área encontram-se os "buffers" para / leitura e escritura do disco fixo. Em particular, nos endereços 40H a 4CH encontra-se o bloco de controle do disco fixo ("DISK CON TROL BLOCK"- DCB), que é usado pela UCP para enviar comandos ao controlador e para ler o estado do controlador. O formato do DCB é dado abaixo:

ENDEREÇO (Hexadecimal)	CONTEÚDO
40	comando
41	" STATUS "

42-43	endereço do "buffer"
44-45	endereço do setor
46-49	-
4A	número de setores para transferência
4B	número de setores alternativos restantes
4C	"CHECKSUM"

Existem cerca de 17 diferentes comandos para o disco. Os comandos utilizados pelo interpretador são: ler um setor, gravar um setor. Após a execução de cada comando, um código é devolvido na posição"STATUS" indicando se houve erro e o tipo do erro. O "CHECKSUM" é um "ou-exclusivo" do conteúdo das posições 40 a 4B, que é verificado pelo controlador do disco para validar o bloco de controle. (Apêndice 7)

O controlador do disco fixo da atual instalação não interrompe a UCP do COBRA-400. O término das operações de Entrada/Saída é feito através de testes ("busy wait") pela UCP da posição "STATUS" do DCB.

Por razões de economia foram usadas como "buffers" de leitura/escrita do disco as áreas de memória correspondentes aos vídeos dos terminais e que estão dentro da área de memória comum ao controlador e à UCP.

4.5.2 - AÇÕES READ E WRITE

As ações READ e WRITE foram implementadas a partir das rotinas de entrada e saída do disco fixo.

A ação READ tem a forma:

READ nome do arquivo,

e a sua função é permitir ao usuário ler um arquivo do disco,cujo conteúdo é um programa BASIC.

A ação WRITE tem a forma:

WRITE nome do arquivo,

cuja função é gravar o programa corrente no arquivo especificado. (O espaço para os arquivos deve ter sido criado previamente através da diretiva "CREATE" do sistema operacional do COBRA-400).

4.5.3 - COMANDOS GET E OUT

Se durante a execução de um programa BASIC,o usuário quisesse analisar os resultados de um problema, o único / meio existente era através da saída no terminal de vídeo. Houve, então, necessidade de comandos que permitissem guardar resultados de programas para posterior análise e processamento. Foram implementados os comandos "GET" e "OUT" com os mesmos parâmetros e funções dos comandos "INPUT" e "PRINT", mudando o periférico para o disco fixo.

A cada terminal foi associado um par de arquivos fixos de entrada e saída. Ao terminal i foram associados os arquivos "INPUT i" e"OUTPUT i".

O comando GET tem a forma:

número de linha GET lista de variáveis,

onde o valor de cada variável é lido do arquivo "INPUT".

O comando OUT tem a forma:

número da linha OUT lista de <expressão> ,

onde <expressão> pode ser expressão aritmética, variável, cadeia de caracteres ou constante numérica, e a função é imprimir seus valores ou cadeias no formato ASCII no arquivo "OUTPUT" do usuário.

CAPÍTULO 5

ESCALONAMENTO DE PROCESSOS

5.1 - INTRODUÇÃO

A rotina do sistema BASIC, responsável pela distribuição das atividades do processador central entre diversos processos, é chamada de "escalador " ("SCHEDULER") de processos . Com cada usuário (terminal) está associado um processo.

A política de escalonamento usada, neste sistema , é do tipo preemptivo circular [7] , ou seja, a interrupção forçada de um processo para que outros processos possam usar o processador. O controle do processador é passado sucessivamente para cada processo "ativo", que recebe uma "porção" de tempo ("time-slice") fixa, antes de ser novamente interrompido, permitindo assim que todos os processos tenham uma distribuição equitativa do tempo do processador. Esta política evita um processo de monopolizar o processador, seja por erros (um laço infinito) ou porque o tempo requerido de execução é muito grande. Esta proteção é feita através do relógio de tempo real do COBRA-400 que interrompe o processador a cada 50 ms.

O processo de um usuário pode estar em cada instante num dos seguintes estados :

(1) - "ativo" ou "pronto para executar"

Neste estado o processo está à espera da liberação do processador, para continuar executando (só passa para o estado 3). O seu contexto imediato encontra-se salvo numa tabela de 10"bytes" na área de dados do processo.

(2) - "à espera de dados" provenientes do teclado.

Neste estado o processo está bloqueado e só sairá do mesmo e colocado em execução (estado 3) quando um caractere especial indicando fim de entrada de dados for enviado pelo teclado.

(3) - "em execução".

Neste estado o processador está executando o processo do usuário. O processo só sairá deste estado quando um dos eventos abaixo ocorrer:

i) expira a "porção" de tempo (passa para o estado 1)

ii) quando a rotina QUITT detecta um CONTROL S. (passa para o estado 2)

iii) na execução do comando INPUT (passa para o estado 2)

iv) quando termina a execução da ação (vai para o estado 2)

A fim de apresentar um grau de interatividade razoável com o usuário, o controle do processador pode também ser transferido do processo de um usuário para outro através de interrupções do teclado, a ser descrito em seguida.

5.2 - TRANSFERÊNCIA DE CONTROLE ATRAVÉS DA INTERRUPÇÃO DO TECLADO

Uma interrupção do teclado é atendida independentemente do processo que tenha o controle do processador. Se ocorre a interrupção no processo A, o sistema BASIC, ao terminar o atendimento desta interrupção, verifica se o caractere enviado é "especial", o que causa uma transferência de controle para o processo B que originou a interrupção.

Os caracteres ditos especiais são:

1 - "NEW LINE" - indica que o usuário terminou a edição de uma linha.

2 - { "CONTROL Y" / "CONTROL S" / "CONTROL Q" } - caracteres de controle do vídeo.

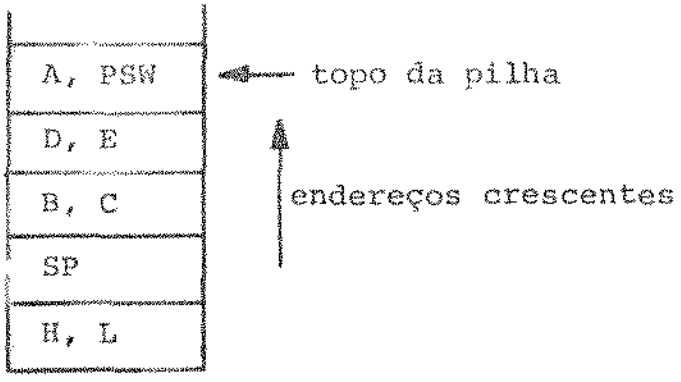
3 - "RUBOUT" - apaga o último caractere enviado.

4 - "ERROR RESET" - apaga a linha que está sendo inserida.

O tratamento desses caracteres difere dos demais por exigir uma resposta imediata do Interpretador, necessária para manter o nível de interatividade com o usuário.

Ao terminar a edição de dados (inserção de uma linha do programa ou entrada de dados para o comando INPUT) o usuário necessariamente envia o caractere especial "NEW LINE", causando a passagem do estado 2 do processo para o estado 3.

O contexto imediato do processo A interrompido é salvo na pilha do sistema BASIC da seguinte forma:



pilha do sistema BASIC

O contexto imediato do processo B (que originou a interrupção) é guardado na tabela de contexto da seguinte / forma:

Tabela que arquiva contexto do programa

Os contextos na pilha do sistema e na tabela do processo B estão em ordem inversa para permitir que a transferência de contexto seja feita de maneira eficiente através das /

instruções PUSH e POP.

A transferência de controle do processo A para o processo B segue o algoritmo abaixo:

CASO (i) - Processo A e processo B distintos

1 - transfere o contexto imediato do processo A da pilha do sistema BASIC para a tabela de contexto imediato do processo A;

2 - determina o valor do registrador base (RB) relativo ao processo B;

3 - restaura o contexto imediato do processo B;

4 - ativa interrupções;

5 - retorna ao processo B;

CASO (ii) - Processo A e processo B são o mesmo processo

1 - restaura o contexto imediato salvo na pilha do sistema BASIC;

2 - ativa interrupções;

3 - retorna

5.3 - INTERRUPÇÃO DO RELÓGIO DE TEMPO REAL

Associado ao relógio existe um contador de tempo que varia de 0 a 9 e é incrementado (módulo 10) a cada interrupção do relógio. Sua finalidade é medir a "porção de tempo" que na implementação é igual a 10^0 ciclos do relógio (0,5 segundo).

O contador é zerado também quando uma interrupção de teclado devolve o controle para um processo (passagem do estado 2 para o estado 3).

Com cada processo existe associado um contador do tempo de espera desde que o processo foi colocado no estado ativo (valor inicial 0). Este contador é incrementado para cada processo "ativo" ao final de cada "porção" de tempo.

A rotina que atende a interrupção do relógio faz o seguinte:

(i) - desliga o pedido de interrupção do relógio.

(ii) - incrementa (módulo 10) o contador do relógio.

(iii) - pisca os cursores dos quatro vídeos.

(iv) - se o contador igual a 10 chama o escalador de processos, caso contrário retorna o controle ao processo interrompido.

A política de escalonamento organiza os processos numa fila circular obedecendo ao seguinte critério para determinar o próximo processo a ser executado:

(i) - processos que estiverem à espera de dados pelo teclado (estado 2) não serão escalados para execução.

(ii) - a prioridade do processo dentro da fila é o seu tempo de espera. O processo que se encontra no estado ativo há mais tempo, isto é, com maior tempo de espera, possui maior prioridade. Se houver mais de um

processo nessas condições o processo associado ao terminal de menor número é escolhido para execução.

A "porção" de tempo deve ser calculada de maneira a não sobrecarregar o sistema devido à frequência de transferência do processador entre os processos ativos, e de modo a dar / tempos de respostas razoáveis para tarefas de curta duração.

Se um processo requer 1 segundo de tempo do processador, o tempo de resposta, caso todos os outros processos estejam ativos (o pior caso) será de 2,5 segundos, isto é, para cada 0,5 segundo de execução o processo espera 1,5 segundos.

O tempo máximo de resposta de um usuário é dado, pois, pela equação linear:

0,5 + (2t - 1) x 2 segundos, onde t é o tempo de processamento requerido em segundos. De modo geral se t_s for a duração / da "porção" de tempo em segundos e n for o número de usuários ativos, o tempo máximo de resposta será dado por ($t \geq t_s$):

$$t_s + \left(\frac{t}{t_s} - 1 \right) \times n \, t_s \quad \text{segundos}$$

Como exemplo, foi medido o tempo de resposta de um programa que resolve um sistema de 5 equações lineares pelo / método de eliminação de GAUSS, para 1, 2, 3 e 4 usuários ativos, obtendo-se:

Nº DE USUÁRIOS ATIVOS	TEMPO DE RESPOSTA(seg)
1	8
2	16
3	24
4	32

É interessante observar nesse teste que as rotinas de escalonamento de processos e de transferência de contexto não sobrecarregam o sistema, usando tempo do processador invisível ao usuário. (Veja apêndice 8, Rotina de Escalonamento de Processos)

5.4 - TRANSFERÊNCIA DE CONTROLE ATRAVÉS DE PROGRAMA

Durante a interpretação das ações "RUN" e "LIST" o processo pode entrar num laço à espera de dados, ao executar o comando INPUT, ou à espera de um "CONTROL Q" na execução da rotina "QUITT".

Para evitar que o processador gaste tempo em "espera ocupada" ("busy wait") para recepção dos dados, o interpretador simula uma interrupção, para salvar o contexto do processo, e chama o escalador de processos colocando o processo interrompido / no estado à espera de dados.

A receber os dados o processo volta a executar a ação "RUN" ou "LIST" interrompida.

CAPÍTULO 6

MEDIDAS DE DESEMPENHO E CONCLUSÕES

6.1 - MEDIDAS DE DESEMPENHO

Para comparar o aumento do tempo de execução entre os sistemas mono e multiusuário foi executado um programa / BASIC que resolve um sistema de equações lineares pelo método de eliminação de GAUSS. (Apêndice C9)

Os tempos de execução obtidos foram:

Nº DE EQUAÇÕES	TEMPO(seg) SISTEMA MULTI	TEMPO(seg) SISTEMA MONO
2	01	01
3	02	02
4	05	04
5	08	07
10	54	48
20	355	314

O aumento do tempo de execução, neste exemplo, do sistema multiusuário em relação ao monousuário foi em média 12,5%. Isto foi possível graças à estrutura do interpretador / que facilitou a substituição das instruções que lidam com endereços

(relocação) por um conjunto máximo de 10 instruções e estas relocações ocorrem um número de 165 vezes. Além disto, as rotinas de ponto-flutuante não utilizam nenhuma dessas instruções.

6.2 - SUGESTÕES PARA FUTURAS EXPANSÕES E CONCLUSÕES

Como futura expansão desse trabalho, sugere-se o desenvolvimento de um sistema "BASIC Comercial" que permite operações de aritmética decimal com precisão estendida no lugar das atuais operações de ponto-flutuante que só admitem 7 algarismos significativos.

A introdução de matrizes, além dos vetores / já existentes, seria outra possível expansão, assim como a introdução de um conjunto de funções científicas (seno, cosseno, tangente, etc.)

Para as aplicações comumente encontradas em BASIC, o sistema multiusuário, operacional desde julho de 1980, apresenta maior versatilidade do que o sistema original devido a uma utilização mais eficiente dos recursos do COBRA-400.

O sistema pode ser de grande utilidade no ensino de introdução à programação. Além disto, pode ser utilizado para o desenvolvimento de aplicações científicas de pequeno porte.

BIBLIOGRAFIA

[1] SAMMET,J.E.- *Programming languages; history and fundamentals.* Englewood Cliffs, Prentice-Hall,1969. p.229-40.

[2] DICKENSON,J.; BARBER,J. & TEETER,J. - Lawrence Livermore Lab's 8080 BASIC. *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, Menlo Park: 8-62, jan. 1977.

[3] COBRA Computadores e Sistemas Brasileiros S/A. *Manual do usuário COBRA-400.* Rio de Janeiro, 1977.

[4] CÉTOLO,M.T.A.- *Manual do utilitário do sistema BASIC-8080.* Campinas, UNICAMP/Lab. de Microprocessadores, 1980.

[5] COBRA Computadores e Sistemas Brasileiros S/A. *COBRA-400 maintenance manual.* Rio de Janeiro, 1976.

[6] INTEL - *8080/8085 Assembly Language Programming manual 9800940. 1978.*

[7] GUIMARÃES, C.C. - *Princípios de sistemas operacionais.* Rio de Janeiro,Ed. Campus, 1980.

[8] COBRA Computadores e Sistemas Brasileiros S/A - *Mini-Disk multi-sector BSH firmware*, Rio de Janeiro, 1977. (Manual 980176)

[9] KOWALTOWSKI,T.- *Implementação de linguagens de programação*. São Paulo, USP/Escola de Computação, 1979. 257p.

APÊNDICE 01

COMANDOS BASIC 8080

COMANDOS	FUNÇÃO
REM	Indica um comentário
END	Fim do programa
STOP	Pára execução do programa
DIM	Declara vetores
LET	Atribui um valor a uma variável
IF-THEN	Desvio condicional
INPUT	Lê dados via terminal
PRINT	Imprime expressões via terminal
GET	Lê dados via disco fixo
OUT	Imprime expressões via disco fixo
GO TO	Desvio condicional
FOR-NEXT	Comando de iteração
GOSUB NN	Transfere controle para uma sub-rotina cujo número de linha é NN
RETURN	Retorna o controle para a linha após o último GOSUB
CALL(N,A,B...)	Transfere o controle para a sub-rotina escrita em linguagem de montagem cujo número é N. A,B... são parâmetros

APÊNDICE 02

VARIÁVEIS DE CONTROLE UTILIZADAS PELO INTERPRETADOR

OBUFF - "buffer" de entrada e saída. A primeira posição contém o número de caracteres do "buffer" (63 caracteres no máximo.

STLINE - apontador para primeira linha do programa corrente a ser interpretada.

NLINE,NL2 } contêm endereço da linha corrente, número da linha,
NL4, NL6 } apontador para próxima linha, número de caracteres.

KLINE, KL2 } idem para linha anterior
KL4, KL6 }

PLINE, PL2 } idem para a próxima linha
PL4, PL6 }

MULT1 ,MULT2 - usados para armazenar valores binários que serão multiplicados.

STSPAC - apontador para a primeira variável da tabela de símbolos.

LPNT - apontador para a linha corrente no momento da execução.

CPNT - apontador para caractere da linha corrente sendo executada.

FREG1, FREG2 - registros em "ponto-flutuante" usados para as operações aritméticas.

HLINP,CREG - memória temporária. Usada na execução do comando INPUT

MODE - Ø se a constante numérica é declarada no programa e 1, caso contrário.

MESCR - área de memória usada pelo comando "CALL" para armazenar constantes ou resultados de expressões aritméticas que são parâmetros.

CTYSQ - indica se a tecla CONTROL Y, CONTROL S ou CONTROL Q foi detectada. É inicializada com zero no início da execução das ações.

SUBS - entrada da tabela de chamada de sub-rotinas em linguagem de montagem.

BOTNS - Base da pilha de "aninhamentos" dos comandos FOR-NEXT.

TOPNS - limite da pilha de"aninhamentos" dos comandos FOR-NEXT. (máximo de 8 "aninhamentos")

NEST - topo da pilha de "aninhamentos"

VARAD, VNAME
VLOC, FLIMT
- área de memória usada pelos comandos FOR-NEXT.

BOE, EOE,
NAR, NARD } contêm informações de um arquivo do disco fixo.

RNARD, RBOE - aponta para o setor do último registro lido de um arquivo.

BUFFER - "buffer" para leitura/escritura das rotinas do disco fixo.

NARQ - nome do arquivo para leitura/escritura das rotinas do disco.

VARIÁVEIS DE CONTROLE GLOBAIS

NLINE - apontador para primeira posição livre de memória (valor inicial 1000H)

STKFULL - indica fim da pilha do usuário (valor = 10H)

RB - apontador para o número de página do usuário corrente.

APÊNDICE 03

MAPA DE MEMÓRIA DO SISTEMA BASIC

ENDEREÇO (H)	CONTEÚDO
Ø	1a página (não utilizada)
ØFF	
1ØØ	Rotinas de Entrada e saída via terminal. Interrupções, "buffers", escalador de processos, Pilha do sistema BASIC
6FF	
7ØØ	Área de memória correspondente aos 4 terminais
9FF	
1ØØØ	Espaço reservado para programa dos usuários e tabelas de símbolos
D85Ø	
D9ØØ	Rotinas de Entrada e saída via disco fixo
DDFF	

ENDEREÇO	CONTEÚDO
DEØØ	Rotinas de ponto-flutuante
E4FE	
E5ØØ	Interpretador BASIC
F6FF	
F7ØØ	Variáveis de controle Usuário 1 Pilha usuário 1
F8FF	
F9ØØ	Variáveis de controle Usuário 2 Pilha usuário 2
FAFF	
FBØØ	Variáveis de controle Usuário 3 Pilha usuário 3
FCFF	
FDØØ	Variáveis de controle Usuário 4 Pilha usuário 4
FEFF	
FFØØ	"ROM"
FFFF	

APÊNDICE 04

RELOCAÇÃO DAS INSTRUÇÕES ESPECIAIS

1 - Instrução LDA adr

ocorrência estática no interpretador = 8 vezes

A - Substituição direta

			Nº DE "BYTES"	Nº DE CICLOS
	PUSH	H	1	11
	LHLD	RB	3	16
	MVI	L,adr	2	10
	MOV	A,M	1	7
	POP	H	1	10
			8	54
		TOTAL	64	432

B - Chamada de rotina

			Nº DE "BYTES"	Nº DE CICLOS
	MVI	A, adr	2	7
	CALL	LDAA	3	17
			5	
LDAA:	PUSH	H	1	11
	LHLD	RB	3	16
	MOV	L,A	1	5
	MOV	A,M	1	7
	POP	H	1	10
	RET		1	10
			8	83
		TOTAL	48	664

2 - Instrução STA adr

ocorrência estática no interpretador = 4 vezes

A - Substituição direta

		Nº DE "BYTES"	Nº DE CICLOS
PUSH	H	1	11
LHLD	RB	3	16
MVI	L,adr	2	10
MOV	M,A	1	7
POP	H	1	10
		8	54
	TOTAL	32	216

B - Chamada de rotina

			Nº DE "BYTES"	Nº DE CICLOS
	PUSH	B	1	11
	MVI	C,adr	2	10
	CALL	STAA	3	17
	POP	B	1	10
	;		7	
STAA:	PUSH	H	1	11
	LHLD	RB	3	16
	MOV	L,C	1	5
	MOV	M,A	1	7
	POP	H	1	10
	RET		1	10
			8	107
		TOTAL	36	428

3 - Instrução LHLD adr

Ocorrência estática no interpretador = 61

A - Substituição direta

		Nº DE "BYTES"	Nº DE CICLOS
PUSH	D	1	11
LHLD	RB	3	16
MVI	L,adr	2	10
MOV	E,M	1	7
INX	H	1	5
MOV	D,M	1	7
XCHG		1	4
POP	D	1	10
		11	70
	TOTAL	671	4270

B - Chamada de Rotina

		Nº DE "BYTES"	Nº DE CICLOS
LXI H, adr		3	10
CALL LHLDA		3	17
:		6	
LHLDA: PUSH	D	1	11
XCHG		1	4
LHLD	RB	3	16
MOV	L,E	1	5
MOV	E,M	1	7
INX	H	1	5
MOV	D,M	1	7
XCHG		1	4
POP	D	1	10
RET		1	10
		12	106
	TOTAL	378	6466

4 - Instrução SHLD adr

ocorrência estática no interpretador = 34 vezes

A - Substituição Direta

		Nº DE "BYTES"	Nº DE CICLOS
PUSH	D	1	11
PUSH	H	1	11
LHLD	RB	3	16
MVI	L,adr	2	10
POP	D	1	10
MOV	M,E	1	7
INX	H	1	5
MOV	M,D	1	7
XCHG		1	4
POP	D	1	10
		13	91
	TOTAL	442	3094

B - Chamada de rotina

			Nº DE "BYTES"	Nº DE CICLOS
	PUSH	D	1	11
	LXI	D,adr	3	10
	CALL	SHLDA	3	17
	POP	D	1	10
	⋮		8	
SHLDA:	PUSH	H	1	11
	LHLD	RB	3	16
	MOV	L,E	1	5
	POP	D	1	10
	MOV	M,E	1	7
	INX	H	1	5
	MOV	M,D	1	7
	XCHG		1	4
	RET		1	10
			11	123
		TOTAL	283	4182

5 - Instruções LXI rp, adr

As instruções desse grupo não foram substituídas através de chamada de rotina porque suas implemetações seriam por demais ineficientes.

5.1 - LXI H,adr

ocorrência estática no interpretador = 48

		Nº DE "BYTES"	Nº DE CICLOS
LHLD	RB	3	16
MVI	L,adr	2	10
		5	26
	TOTAL	240	1248

5.2 - LXI B, adr

ocorrência estática no interpretador = 1

		Nº DE "BYTES"	Nº DE CICLOS
PUSH	H	1	11
LHLD	RB	3	16
MOV	B,H	1	5
MVI	C,adr	2	10
POP	H	1	10
		8	52
	TOTAL	8	52

5.3 - LXI SP,adr

ocorrência estática no interpretador = 3

		Nº DE "BYTES"	Nº DE CICLOS
PUSH	H	1	11
LHLD	RB	3	16
INR	H	1	5

MVI	L,adr	2	10
SPHL		1	5
POP	H	1	10
		9	57
	TOTAL	27	171

5.4 - LXI D,adr

ocorrência estática no interpretador = 6

		Nº DE "BYTES"	Nº DE CICLOS
PUSH	H	1	11
LHLD	RB	3	16
XCHG		1	5
MVI	E,adr	2	10
POP	H	1	10
		8	52
	TOTAL	48	312

APÊNDICE 05

ROTINA DE INTERRUPÇÃO

```
INTER:  SHLD   STSI          ; salva contexto imediato
        LXI    H,Ø           ; na pilha do sistema
        JNC    CYØ
        DAD    SP            ; salva SP sem destruir "carry bit"
        STC
        JMP    INTO
CYØ:    DAD    SP
INTO:   LXI    SP,STSI       ;(SP) aponta para pilha do sistema.
        PUSH   H
        PUSH   B
        PUSH   D
        PUSH   PSW
        IN     Ø             ; (A) ← nível do periférico
        CPI    1Ø            ; Se (A) = 1Ø
        JNZ    CLOCK         ; então Interrupção do Teclado
        IN     3EH           ; (A) ← nº do terminal (0,1,2 ou 3)
        MOV    B,A
        OUT    3EH           ;desliga pedido de interrupção
        IN     3FH           ; (A) ← código da tecla
        CALL   KYCV          ; rotina que converte código
        MOV    C,A           ; em caractere ASCII. (C) ← caractere
        CALL   BUFFER        ;Rotina que insere caractere no "buffer"
                             ; e ecoa no vídeo
        MOV    A,C           ; verifica se caractere especial
        CPI    1EH           ; Se (A) = "Rubout" então chama
        JZ     TRANSF        ; rotina que transfere controle
                             ; do processador
```

```
        CPI   ØDH          ; se (A) = "New line" então chama
        JZ    TRANSF       ; rotina que transfere o controle
        ANI   ØFH          ; se (A) = "um dos caracteres de controle
                           ; do vídeo"
        CPI   ØFH          ; então chama rotina que transfere o con
                           ; trole
        JZ    TRANSF       ; senão
SAME:   POP   PSW          ; processo que interrompeu é o mesmo que
        POP   D            ; terá o controle
        POP   B            ; restaura contexto imediato
        POP   H
        SPHL
        LHLD  STSI
        EI                 ; ativa interrupçãp
        RET                ; retorna
CLOCK:  XRA   A            ; senão interrupção do relógio
        OUT   2            ; desliga pedido de interrupção
        LDA   CONT         ; se (CONT) < 10
        INR   A            ; então
        CPI   10           ;
        JZ    CLCK2        ; (CONT) ← (CONT) + 1
        STA   CONT         ; retorna o controle para o processo in-
        JMP   SAME         ; terrompido
CLCK2:  XRA   A            ; senão
        STA   CONT         ; (CONT) ← Ø
        CALL  PISCA        ; chama rotina que pisca     os cursores
        JZ    ESCAL        ; chama escalador de processos (ver apên-
                           ; dice 8)
```

APÊNDICE 06

ROTINA DE ENTRADA DE DADOS DO INTERPRETADOR

```
          ;(A) = Nº do usuário que pede o dado

ENTRADA : MOV  B,A          ;(B) ← nº do usuário
          LXI  H,CBØ        ; cálculo dos apontadores
          ADD  L            ; CB e FB usando o nº do usuário como
          MOV  L,A          ; índice numa tabela
          LXI  D,FBØ        ;
          MOV  A,B          ;(HL) ← CB
          ADD  E            ;(DE) ← FB
          MOV  E,A          ;
DEL:      DI                ;início da região crítica
          MOV  A,M          ; (A) ← (FB)
          XCHG              ;
          CMP  M            ; Se (CB) ≠ (FB)
          EI                ; FIM da Região Crítica
          JZ   UNDER        ; então
          DI
          CPI  3FH          ; se (CB) = 3FH
          JNZ  DELT2        ;
          XRA  A            ; então (CB) ← Ø
          MOV  M,A          ;
          JMP  DELT3
DELT2:    INR  A            ; senão (CB) ← (CB) + 1
          MOV  M,A          ;
DELT3:    EI
          LXI  H,BUFFØ      ; cálculo do"buffer"correspondente ao
          ADD  L            ; usuário
DELT5:    DCR  B            ;
```

```
        JM    DELT4          ;
        ADI   40H
        JMP   DELT5          ;
DELT4:  MOV   L,A            ; (HL) ← "buffer"
        MOV   A,M            ; (A) ← (Buffer (CB))
        RET                  ; retorna
UNDER:  RST   4              ; senão  "buffer vazio"
        JMP   DEL            ; chama escalador de processos.
```

APÊNDICE 07

FORMATO DO DISCO FIXO DO COBRA-400

A - CARACTERÍSTICAS FÍSICAS

CAPACIDADE DE ARMAZENAMENTO	5.3 ou 10.6"M Bytes"
Nº DE CILÍNDROS	2
Nº DE TRILHAS	400
Nº DE BYTES/SETOR	512
Nº DE SETORES/TRILHA	13 ou 26
TEMPO DE REVOLUÇÃO	25ms
TEMPO MÉDIO DE POSICIONAMENTO DO BRAÇO "SEEK"	50ms 70ms

B - FORMATO DO DIRETÓRIO DO DISCO

SETOR	CONTEÚDO
0 a 1	"MINI-DISK LOADER"
3 a 32	" HEADER"
32 a 83	Entradas do diretório
Demais	Arquivos e espaços disponíveis

C - FORMATO DO DIRETÓRIO DE UM ARQUIVO

POSIÇÃO (Hexadecimal)	CONTEÚDO
0	Tamanho de entrada (40H)
1-8	Nome do arquivo
9	-
A-B	Tipo do arquivo
C-D	Nº do setor inicial do arquivo (BOE)
E-F	Nº do setor final (EOE)

10-11	Nº do último setor lido ou gravado (CDA)
12-13	Deslocamento no setor em "bytes" (CDAR)
14-15	Nº do setor com espaço disponível (NAR)
16-17	Indica 1º "byte" disponível (NARD)
18-19	Tamanho do registro
Demais	Não utilizados

APÊNDICE 08

ROTINA DE ESCALONAMENTO DE PROCESSOS

```
ESCAL:  MVI   C,4          ; (C) ← nº total de processos
        MVI   B,ØFFH       ;
        LXI   H,MODØ       ;
ESCL1:  MOV   A,M          ; (A) ← estado do processo
                           ; (A) = 1ØH ≡ estado 1 ou "ativo"
                           ; (A) = 2ØH ≡ estado 2 ou "à espera de dados"
                           ;
        CPI   2ØH          ; Se processo está no estado 2
        JZ    ESCL3        ; então
ESCL2:  INX   H            ; verifica estado do próximo processo
        DCR   C            ; até (C) = Ø.
        JNZ   ESCL1        ;

                        ;Neste ponto os estados de todos os processos
;foram verificados , os tempos dos processos "ativos" incrementados e
;(B) = número do processo de maior tempo de espera( se todos no esta-
;do 2 então(B)= FFH)

        INR   B            ; se (B) = FFH
        JZ    SAME         ; então devolve o controle para processo
        SUB   B            ;  interrompido
        MOV   B,A          ; senão chama rotina
        JMP   TRANSF       ; que transfere o controle do processador.
ESCL3:  MVI   A,4          ; senão processo i está "ativo"
        SUB   C            ; (A) ← nº do processo i
        PUSH  H            ;
        LXI   H,TEMPØ      ; cálculo do tempo de espera
        ADD   L            ; correspondente ao processo i
```

```
        MOV   L,A          ; (HL) = Tempoi
        INR   M            ; (Tempoi ) ← (Tempoi) + 1
        MOV   D,M          ; seleciona processo de tempo
        MOV   A,B          ; de espera maior
        CPI   ØFFH         ; se (B) = ØFFH
        JNZ   COMP         ; então nenhum processo foi ainda selecio
                           ; nado        e
        MOV   B,C          ; (B) ← nº do processo i
        MOV   E,D          ; (E) ← Tempo de espera do processo i
        POP   H            ;
        JMP   ESCL2        ; senão
COMP:   MOV   A,D          ; se tempo de espera do processo
        CMP   E            ; salvo em B for maior que tempo
        JNC   ESCL4        ; de espera do processo i
        POP   H            ; então  vá para ESCL2
        JMP   ESCL2        ;
ESCL4:  MOV   B,C          ; senão (B) ← nº do processo i
        MOV   E,D          ; e (E) ← Tempo de espera do processo i
        POP   H            ;
        JMP   ESCL2        ;
```

APÊNDICE 09

PROGRAMA QUE RESOLVE UM SISTEMA DE EQUAÇÕES LINEARES PELO MÉTODO DE GAUSS

```
ØØØØ REM "PROGRAMA PRINCIPAL"
ØØ1Ø DIM A(401),B(21), X(21)
ØØ2Ø INPUT N
ØØ3Ø FOR I = 1 TO N*N
ØØ4Ø GET A (I)
ØØ5Ø NEXT I
ØØ6Ø FOR I = 1 TO N
ØØ7Ø INPUT B (I)
ØØ8Ø NEXT I
ØØ9Ø REM "CHAMADA DE TRIÂNGULO"
Ø1ØØ GOSUB 2ØØØ
Ø11Ø REM " CHAMADA DE GAUSS"
Ø12Ø GOSUB 1ØØØ
Ø13Ø FOR I = 1 TO N
Ø14Ø PRINT X(I)
Ø15Ø NEXT I
Ø16Ø STOP

1ØØØ REM "ROTINA GAUSS"
1Ø1Ø LET X(N) = B(N) / A(N*N)
1Ø2Ø FOR J = N-1 TO 1 STEP -1
1Ø3Ø LET S = Ø
1Ø4Ø LET J1 = J1 -1
1Ø5Ø LET J1 = J1*N
1Ø6Ø FOR K = J + 1 TO N
1Ø7Ø LET S1 = X(K) * A(J1 + K)
1Ø8Ø LET S = S + S1
1Ø9Ø NEXT  K
```

```
11ØØ LET X1 = B(J) -S
1110 LET X(J) = X1/A(J1 + J)
112Ø NEXT  J
113Ø RETURN

2ØØØ REM "ROTINA TRIANG"
2Ø1Ø FOR I =  1 TO N-1
2Ø2Ø LET I1 = I
2Ø3Ø LET I2 = I-1
2Ø4Ø LET I2 = I2 * N
2Ø5Ø LET I3 = I2
2Ø6Ø LET I4 = I3 + I
2Ø7Ø REM "CHAMADA DA ROTINA PIVÔ"
2Ø8Ø GOSUB 3ØØØ
2Ø9Ø LET K = I1
21ØØ IF K = I THEN 212Ø
21Ø9 REM "CHAMA ROTINA TROCA"
211Ø GOSUB 4ØØØ
212Ø FOR J = I+1 TO N
213Ø LET J1 = J - 1
214Ø LET J1 = J1 * N
215Ø LET C = A(J1 + I)/A(I4)
216Ø FOR K = 1 TO N
217Ø LET A1 = C* A(I3 + K)
218Ø LET L = J1 + K
219Ø LET A(L) = A(L) - A1
22ØØ NEXT K
221Ø LET B1 = C * B(I)
222Ø LET B(J) = B(J) -B1
223Ø NEXT J
224Ø NEXT I
225Ø RETURN
```

```
3000 REM "ROTINA PIVÔ"
3010 FOR J = I + 1 TO N
3020 LET J1 = J - 1
3030 LET J1 = J1 * N
3040 IF A(J1 + I) < = A(I2+ 1) THEN 3080
3050 LET I1 = J
3060 LET I2 = I1 - 1
3070 LET I2 = I2 * N
3080 NEXT J
3090 RETURN
4000 REM "ROTINA TROCA"
4010 LET K2 = K
4020 LET K = K - 1
4030 LET K = K * N
4040 FOR K1 = 1 TO N
4050 LET L = I3 + K1
4060 LET M = K + K1
4070 LET T = A(L)
4080 LET A (L) = A(M)
4090 LET   A(M) = T
4100 NEXT K1
4110 LET T = B(K2)
4120 LET B(K2) = B(I)
4130 LET B(I) = T
4140 RETURN
```