UNIVERSIDADE DE SAO PAULO
INSTITUTO DE FISICA E QUIMICA DE SAO CARLOS
DEPARTAMENTO DE FISICA E CIÊNCIA DOS MATERIAIS

Uma proposta de linguagem geradora de imagens em impressoras de página.

Mário Antonio Stefani

Dissertação apresentada ao Instituto de Física e Química de São Carlos, USP para obtenção do título de Mestre em Física Aplicada.

Orientador: Prof.Dr.Jan F.W.Slaets

São Carlos - São Paulo

1990

/2/

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA E QUÍMICA DE SÃO CARLOS

MEMBROS DA COMISSAO JULGADORA DA DISSERTACAO DE MESTRADO DE MARIO ANTONIO STEFANI APRESENTADA AO INSTITUTO DE FISICA E QUIMICA DE SAO CARLOS,DA UNIVERSIDADE DE SAO PAULO,EM 16 DE AGOSTO DE 1990.

COMISSAO JULGADORA:

Prof.Dr.Jan F.W.Slaets

Prof.Dr.Carlos Antonio Ruggiero

Prof.Dr.Edson dos Santos Moreira

Cx. Postal, 369 - FONE (0162) 71-1016 - CEP 13.560 - São Carlos - SP - Telex 162374 - FQSC - BR - BRASIL

Aos meus pais, irmã
e à Te

RESUMO

Uma compacta linguagem descritora de páginas, destinada a impressoras não-impacto de estrutura "raster", é apresentada.

Tal linguagem foi implementada usando o Processador Gráfico TMS34010, da Texas Instruments e possui uma estrutura muito similar à encontrada nas linguagens interpretativas encadeadas. A linguagem é totalmente modular e interativa, e se utiliza de um modelo gráfico simples, visando simular as tarefas normalmente encontradas nas artes tipográficas.

São efetuadas comparações com outras linguagens comerciais, visando avaliar suas possibilidades.

Uma pequena introdução à tecnologia das impressoras laser é apresentada.

ABSTRACT

A small Page Description Language intended for raster non-impact printers is presented.

The language is implemented using the Texas Instruments TMS34010 Graphics System Processor and its structure is similar that encountered in threaded interpretive languages. The language is fully modular and interactive, and uses a simple graphic model to simulate the same common tasks encoutered in typographical arts.

Comparison are made with other commercial languages to perform some evaluations on its possibilities.

A small introduction on the laser printer technology is presented.

AGRADECIMENTOS

Ao Prof. Jan Slaets pelo incentivo, apoio e orientação ao trabalho.

A Sra. Rita de C.P. Sorensen, pela paciência e dedicação na digitação e correção dos manuscritos.

Ao Sr. João Carlos Fantato, pelos desenhos e diagramas.

A Opto Eletrônica S/A, seus funcionários e diretores, e a todos aqueles que, de uma forma ou de outra permitiram a realização deste trabalho.

INDICE

1)- INTRODUÇÃO GERAL

1.1)- Objetivo do trabalho

O presente trabalho trata sobre as necessidades computacionais de impressoras, especificamente as de não impacto e propõe a implementação de uma linguagem controladora . A linguagem proposta é destinada à criação das imagens a serem impressas sendo , basicamente , residente no mecanismo impressor.

A definição da imagem é feita usando comandos e estruturas de alto nível, enviadas ao tal dispositivo. Tais dispositivos são de estrutura preferencialmente "Raster", geralmente de alta resolução e velocidade, imprimindo "páginas" ao invés de "caracteres" ou "linhas".

A linguagem proposta pertence a uma nova classe de linguagens denominadas PDL, de "Page Description Languages" e constituem um sistema capaz de recriar e simular as atividades normalmente executadas no processo de composição tipográfica de uma página a ser impressa. As PDL não devem ser confundidas com os processadores de texto ou editores, apesar de muitas PDL serem extensas o suficiente para abrigar certas funções normalmente encontradas nestes.

As PDL surgiram com o advento das impressoras "não impacto" , principalmente a laser, onde a velocidade e resolução permitiam a criação de páginas impressas, com qualidade comparável aos sistemas de fotocomposição. Sua origem, no entanto, remonta as primeiras impressoras de impacto usadas em computadores.

As primeiras PDL eram constituídas, simplesmente, de algumas funções implementadas no controlador da impressora, que permitiam algumas operações especiais, tais como: sublinhado, troca da forma dos caracteres, etc. O controlador da impressora dividia suas atividades de controle do processo físico de impressão, como acionar agulhas e motores, com a de interpretar certos "caracteres de controle", inseridos no meio do texto transmitido até ele.

Com a popularização da computação gráfica, os dipositivos impressores sofreram a necessidade de se sofisticarem, de forma a permitir maior flexibilidade, culminando no surgimento das impressoras gráficas, onde um conjunto maior de caracteres de controle eram aceitos e permitiam a recepção e impressão de imagens prontas transmitidas até ele.

As imagens foram crescendo de tamanho e resolução, tornando proibitiva a transmissão "bit a bit". A impressora passou a ter um conjunto de comandos que permitiam a transmissão compactada de imagens. Tal conjunto de comandos cresceu tanto que, em uma impressora, o processo físico de impressão ficava a cargo de um controlador e um outro era especialmente designado para interpretar e executar tais comandos.

A capacidade dessas impressoras em qualidade, resolução e velocidade cresceu muito, principalmente com o advento de novas tecnologias, tais como: a eletrofotográfica, ionográfica, etc.

Tais tecnologias permitiram um aumento considerável da resolução, tornando necessário que o mecanismo impressor pudesse receber as imagens a serem impressas na forma de comandos e funções definidoras ao invés de somente a imagem pronta e compactada. Logo visualizou-se a possibilidade de se poder compor páginas, a serem impressas com a mesma qualidade e quantidade de recursos ,somente até então, disponíveis nos sistemas de composição tipográfica. Surgiram então os PDL e os "Desktop Publishing Systems", que possibilitaram aos usuários dos sistemas computacionais criar, por eles próprios, seus documentos e impressos com qualidade comparável aos obtidos com os grandes sistemas de impressão.

No momento, existem vários sistemas PDL, tais como: o Postscript, o Interpress, HPL-Laserjet Command Set, o DCF-IBM.

Com a exceção do Interpress do Xerox e o DCF da IBM, todos os outros citados foram criados para sistemas de pequeno e médio porte, tais como impressoras laser pessoais.

Geralmente, tais linguagens foram escritas para microprocessadores de uso geral. Como a criação e montagem de uma página requer o uso de comandos e estruturas de processamento complexos, geralmente o processamento é lento, pois tais microprocessadores, normalmente, não tem uma arquitetura que os possibilite executar funções gráficas de forma rápida.

No momento, existem certos microprocessadores que possuem arquitetura dedicada a operações gráficas, tal como o TMS34010 da Texas Instruments.

A proposta do autor é a de definir uma linguagem PDL para o TMS34010. A utilização desse processador, devido a sua arquitetura dedicada, permite uma grande facilidade de implementação de certas funções exigidas pelas PDL . Tal fato pode induzir, de certa forma, a adoção de modelos e técnicas não convencionais de lingüística computacional.

Como observado em alguns PDL, o modelo gráfico, ou "filosofia", que os comandos e estruturas seguem ,visam a simular os meios físicos reais que a indústria gráfica convencional utiliza, tais como: "stencil", "pincel", "máscara". A adoção desses modelos visa a tornar o sistema similar e de fácil assimilação pelo usuário, já que os processos são intuitivos fisicamente.

O autor pretende apresentar uma proposta de implementação , composta de uma estrutura lingüística simples , compacta e flexível, sintonizada de forma a explorar ao máximo as características de um processador gráfico adotado, aliando a um modelo gráfico aprasível e de fácil assimilação ,as funções e requisitos que uma linguagem PDL deve apresentar.

1.2) Origens do Trabalho

O presente trabalho tem sua origem no projeto entitulado "Nacionalização de uma Máquina Gravadora a Laser" , efetuado durante os anos de 1984 e 1985, no Centro de Mecânica Fina da EESC-USP, com os fundos oriundos do então FIPEC-Banco do Brasil. Tal projeto visava a nacionalização de um sistema destinado a marcação ("engraving"), utilizando-se de um laser de Nd-YAG, de 60W, controlado por um sistema computadorizado. Neste trabalho , foi de responsabilidade do autor a elaboração e implementação do hardware e software de tal sistema computadorizado. Na época , o autor adotou um microcomputador APPLE II de 64kbytes, onde uma placa de interface D-A de 12 bits acionava diretamente os espelhos galvanômetros, situados na cabeça do laser, que defletiam o feixe deste. O software foi escrito em TRANSFORTH e Assembler 6502. Tal Software foi chamado de SGL (Sistema de Gravação a Laser) e foi apresentado na Feira de Informática SUCESU'85 (Stefani,85). Com a exceção do laser , o sistema foi nacionalizado e o relatório final foi apresentado ao FIPEC, em outubro de 1985.

O software, como foi criado, simulava as funções e possibilidades de um "plotter", podendo o usuário criar formas, figuras e textos, comparáveis ao obtido por esses equipamentos.

A SEI-CTI de Campinas , tomando o conhecimento de tal projeto finalizado, procurou o CEMEFI-EESC-USP para trabalhar num projeto semelhante de nacionalização . Tal projeto , entitulado "Estudo de Viabilidade para a Nacionalização de uma Impressora Laser" era encomendado pela firma EXPANSÃO INFORMATICA, do Rio de Janeiro. Este projeto visava o estudo de viabilidade técnico/econômico para a nacionalização progressiva de uma impressora laser de médio porte . Tal projeto teve início em novembro de 1985 . Foi constituído uma equipe mista de profisionais do CEMEFI , SEI-CTI e da EXPANSÃO , sendo de responsabilidade do autor e de mais três profissionais do CEMEFI o estudo técnico. Foram importadas duas impressoras laser, marca DATAPRODUTCS , que foram inteiramente desmontandas e ensaidas , seus componentes mecânicos e circuitos eletrônicos inteiramente levantados, bem como os softwares de controle. Tal estudo foi concluído em novembro de 1986 onde se verificou que um caminho técnico e economicamente viável de obter um "engine" laser nacional, era o de se modificar uma máquina copiadora de boa qualidade fabricado no país, modificando seus sistemas e controles eletrônicos e adicionando o módulo ótico(Stefani,86).

Terminado o estudo de viabilidade , a segunda face seria a da contrução de 5 protótipos , porém a EXPANSÃO entrou em grave situação financeira , abandonando o projeto.

Nesse interim , a OPTO ELETRONICA S/A de São Carlos , obteve um financiamento do FINEP para desenvolver um protótipo de impressora laser nacional . O autor foi convidado a participar da equipe, que logo decidiu acatar as conclusões do estudo de viabilidade que o mesmo tinha participado(Stefani,88). Com os recursos do FINEP , foram adquiridas algumas máquinas copiadoras nacionais e iniciou-se a modificação de tais equipamentos em junho de 1987. Em maio de 1988, a equipe já tinha concluído a modificação de todo o sistema de controle de impressão ,

iniciando-se a implantação do módulo ótico e do processador de imagens . Nesta época , de comum acordo com o Sr. orientador , ficou decidido que o autor , responsável pela implementação do processador de imagem no protótipo, utilizaria parte deste trabalho como sua tese .

1.3) Organização do Trabalho

O presente trabalho visa a descrever a implementação proposta , comentando sua estrutura e funcionamento.

Primeiramente será feita uma introdução ao funcionamento de impressoras laser , visto que tal tipo de equipamento é o que mais se utiliza das PDL existentes.

O conhecimento do funcionamento da impressora é vital para a confecção das rotinas que interfaceiam com o mecanismo impressor. Essas rotinas, "dependentes da máquina", devem ser capazes de transformar a imagem da memória em sinais adequados para a impressão.

Uma introdução aos PDL mais comuns também será efetuada através exemplos comentados , o que permitirá a visualização das características desejadas de tal classe de sistemas.

Logo após, será discutido o modelo gráfico que será adotado , pois tal modelo define em grande parte as estruturas e comandos que a linguagem deve apresentar.

Em seguida, será apresentado a estrutura básica do funcionamento da linguagem, bem como sua relação com o processador adotado . As estruturas gráficas serão discutidas a seguir, através de exemplos comentados , já explorando as características desejadas de um PDL. Testes de avaliação serão feitos em um sistema físico simulando uma impressora , onde conclusões serão obtidas, junto com sugestões para o aperfeiçoamento do trabalho e sua continuidade futura.

Nos apêndices constarão a lista de comandos da linguagem , bem como a listagem completa de sua implementação para o TMS34010.

II)- UMA INTRODUÇÃO AS IMPRESSORAS LASER

II-1)- Introdução

Neste capítulo será apresentado um breve histórico e também uma introdução ao funcionamento das impressoras laser. Para um maior aprofundamento, recomenda-se a leitura das referências indicadas.

As impressoras laser são caracterizadas como impressoras não-impacto, para diferenciá-la dos processos normais onde a tinta é transferida por processos mecânicos.

Existem uma grande variedade de processos não-impacto (Myers, 84),(Liebermann, 84), (Teja, 85), porém as impressoras laser tem obtido uma grande popularidade nos últimos anos, devido a sua ótima relação custo/benefício. São impressoras rápidas e silenciosas, de alta qualidade e resolução (Merrit, 85), (Hall, 82).

Quando equipados com uma PDL poderosa, são capazes de imprimir com qualidade e potencialidade equivalente aos sistemas de fotocomposição de grande porte.

O processo de impressão laser é uma consequência natural do desenvolvimento do processo mais conhecido como xerografia, que por sua vez é um processo eletrofotográfico.

A invenção do processo eletrofotográfico remonta de 1.938, quando Chestes F. Calson o patenteou em sua forma básica (Schaffert, 78) . O nome advém da forma que uma imagem é produzida, através de um processo elétrico. Neste processo uma superfície fotocondutora é carregado eletrostáticamente. A imagem desejada é, então, projetada sobre a superfície. As regiões onde há incidência de luz, são descarregadas e onde não, permanecem carregadas. Tais regiões, ainda carregadas, atraem partículas de tinta, que serão transferidas ao papel. Dessa forma, regiões onde houver luz, no papel ficarão sem tinta e vice-versa.

A partir das primeiras experiências, até o surgimento da primeira máquina comercialmente disponível, foram gastos 12 anos de pesquisas e investimentos no aprimoramento do processo . A primeira máquina era totalmente manual, o usuário tinha que efetuar 27 operações sequencialmente, e era destinado a produção de "masters" para o sistema "offset" de impressão.

A primeira máquina automática e destinada a utilização como "copiadora", da forma como hoje conhecemos , surgiu em 1.959, pela Haloid Corp. mais tarde conhecida por Xerox Co. O processo se popularizou surgindo variações do método. Também começou-se o estudo da possibilidade de se utilizar o processo eletrofotográfico como método de impressão para computadores. Em 1968, a Xerox mostrou o "Xeronic", uma impressora onde a imagem era produzida por um CRT de alta potência focalizado sobre a superfície fotocondutora, e em 1973 a primeira impressora eletrofotográfica para computador, a Xerox 1200, onde os caracteres eram formados a partir da projeção de máscaras selecionáveis (Rothgordt, 82).

Nessa época surgiram os primeiros lasers e os primeiros estudos de sua utilização no processo eletrofotográfico ocorreram no final da década (Tam,82). A primeira impressora laser, próprialmente dita, foi apresentada em 1975, a IBM 3800, já com capacidade de imprimir 160 páginas/minuto , com resolução de 60000 pontos por polegada quadrada (Starkweather, 80). Logo a seguir, surgiram as máquinas da Xerox, Siemens e Hitachi com desempenhos similares. Todas, devido ao seu alto custo, eram destinadas a grandes sistemas, e exigiam desde sofisticadas organizações corporativas, de forma a garantir o regular suprimento de papel, até sistemas computacionais especialmente dedicados a explorar suas capacidades (Walter, 78), (Elzinga, 81).

Logo em 1976 a IBM lançou o modelo 6670 de menor porte, visando a utilização em organizações de médio porte , seguido da HP e Xerox.

Em 1979 surgia a primeira impressora a laser de pequeno porte e incorporando a tecnologia dos lasers semicondutores pela Canon, com sua LBP-10. Usando o módulo Canon, a HP lançou a HPLaserjet que marcou a revolução do mercado em 1.984, junto com a explosão dos micros pessoais (Catalano,85), (Mayer,87), (Williams,85). Logo em seguida, surgia a Apple Laser-Writer, também utilizando o módulo Canon, porém dotada da primeira linguagem PDL destinada aos "Desktop Publishing System", (Dickinson, 85), (Rosemberg, 85).

Essa linguagem, o PostScript, é baseado na experiências de seus autores, quando desenvolveram os PDL de grande porte para máquinas Xerox, sendo atualmente uma das linguagens mais populares.

II-2)- Principais partes de uma impressora laser

Neste capítulo, será apresentada uma introdução ao funcionamento de uma impressora a laser, descrvendo suas principais partes.

Na figura II-1, abaixo, é apresentado um diagrama de blocos simplificado de uma impressora a laser.

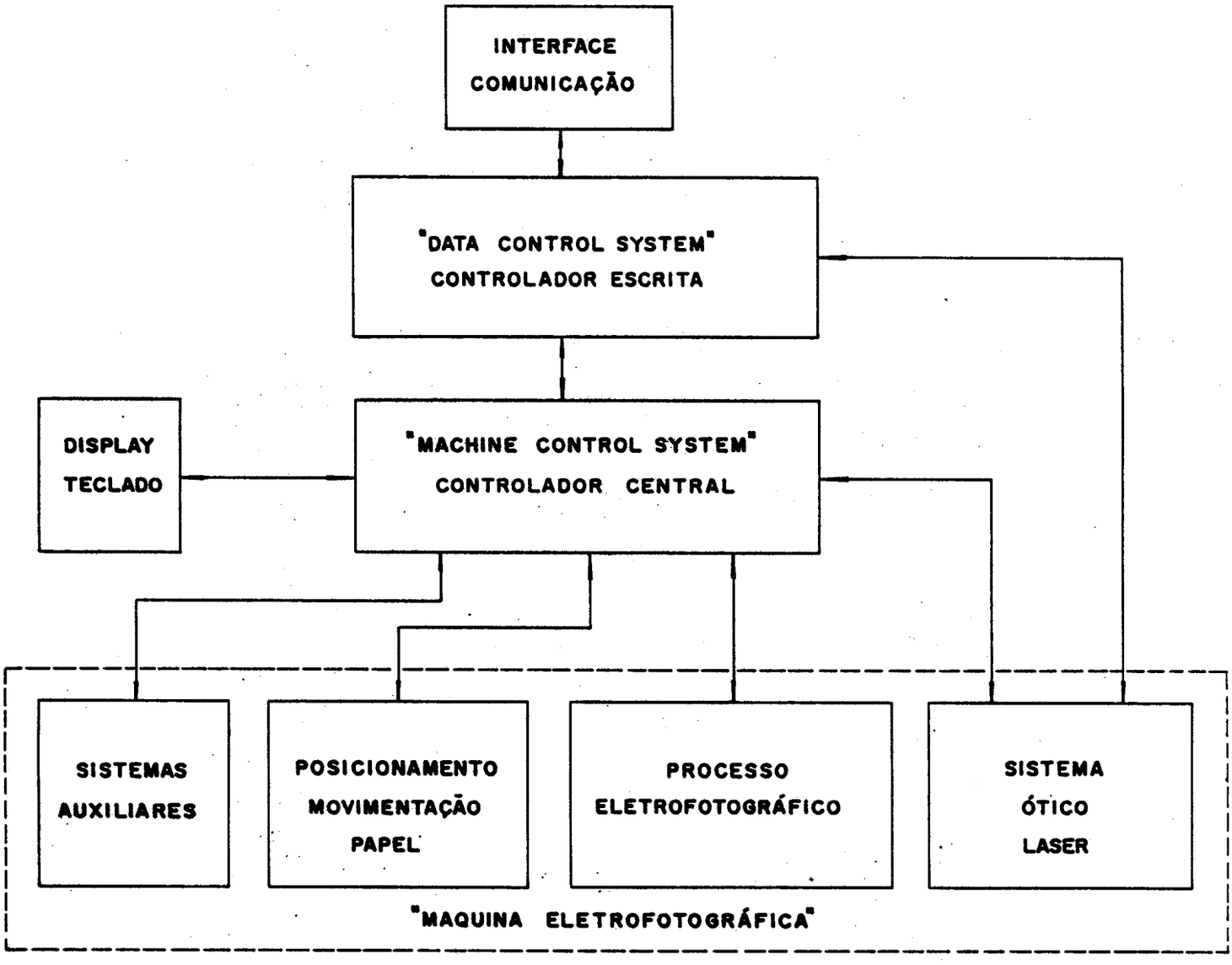


Fig.II-1)- Estrutura geral de uma impressora laser

Basicamente , tem-se os seguintes blocos:

a)- a interface de comunicação é a responsável pela troca de informações entre a máquina e o computador, ou sistemas de onde é o periférico. Normalmente é capaz de receber dados a taxas elevadas, inclusive de multiplas fontes, configurando-os de forma adequada à interpretação pelo controlador de escrita "DCS".

b)- o "Data Control System", ou também controlador de escrita, interpreta e processa os sinais recebidos, sendo o bloco responsável pela geração, na memória, da imagem da página a ser impressa. Este envia os sinais necessários para a modulação do laser e recebe sinais do "MCS" sobre as condições da máquina.

c)- o "Machine Control System", ou controlador central, é o responsável pelo monitoramento e manutenção dos diversos processos físicos necessários a obtenção do papel impresso. O "MCS" controla a movimentação do papel, monitora o processo eletrofotográfico, recebe dados do operador via teclado , envia sinais de configuração e funcionamento ao Display e ao DCS, e atua no sincronismo entre o "DCS" e o sistema laser.

Devido ao grande número de operações necessárias para a efetivação da página a ser impressa, geralmente o DCS e o MCS são atribuídos a processadores distintos e especializados.

A máquina eletrográfica é um conjunto muito grande e complexo de sub-sistemas, que exigem uma grande capacidade de processamento, como será visto a seguir.

II-3)- O processo de impressão.

O processo de impressão utilizado, normalmente, pelas impressoras a laser é o eletrofotográfico, similar às presentes nas máquinas copiadoras "secas", tipo xerox. O processo é ligeiramente modificado, visando principalmente rapidez e qualidade de impressão , bem como para permitir o melhor casamento entre as diversas partes componentes (Lee, 84).(Vahtra,78)

Na figura II-2, é mostrado um esquema simplificado dos estágios necessários para a geração da página impressa. Tal sistema é o utilizado pelas impressoras Dataproducts, com velocidade de 25 páginas/minuto.

De acordo com a figura II-2, pode-se seguir a sequência abaixo:

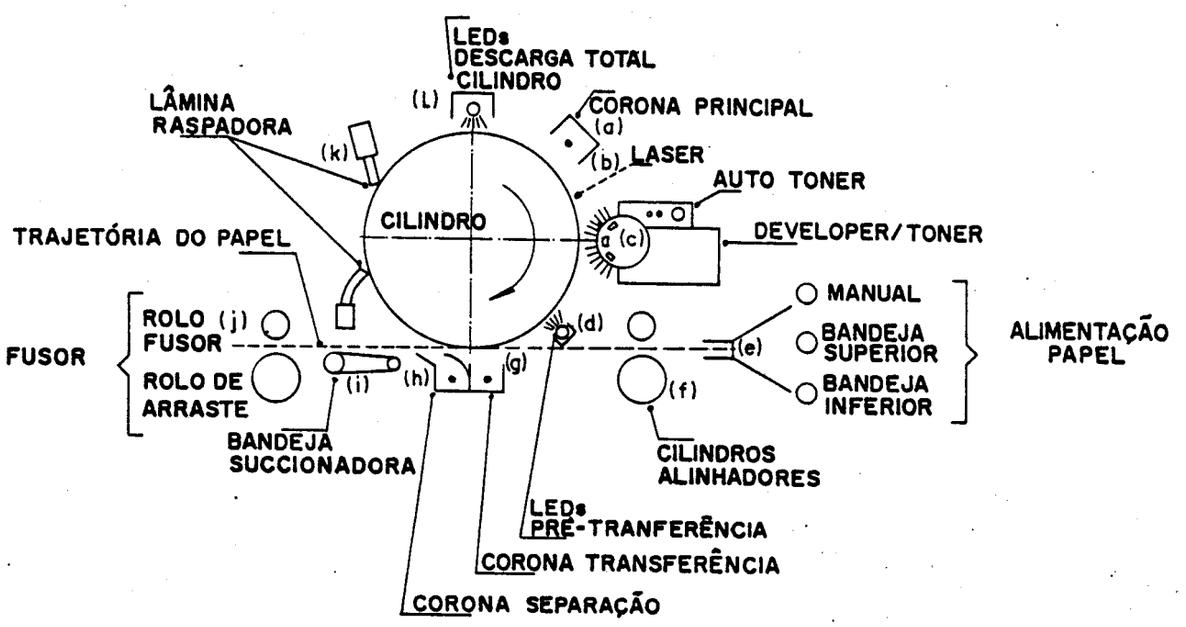


Fig.II-2)- Processo de impressão

1o.)- um dispositivo corona (a), chamado corona principal, polarizado ao redor de +- 5500V e posicionado acima do cilindro fotocondutor, cria cargas negativas na superfície deste. Como o ambiente ali não está iluminado, a camada fotocondutora do cilindro possue alta resistência, impedindo a dissipação dessas cargas;

2o.)- o feixe laser varre a superfície desse cilindro em (b), no sentido axial, modulado com a informação necessária para a formação da imagem desejada. Nota-se que o laser faz a varredura no sentido axial, sincronizado com o movimento relativo do cilindro, produzindo, assim, um padrão "Raster" de geração da imagem. Um ponto onde o laser aceso incide, torna a superfície do cilindro condutora, dissipando as cargas ali localizadas. Onde o laser não incide permanecem as cargas;

3o.)- o toner é carregado positivamente, sendo atraído pelas regiões que ainda permaneceram carregadas, tornando a imagem visível. Esta carga positiva dá-se tanto por polarização do módulo revelador quanto pelo atrito e agitação com o pó magnético (" developer"), que serve de veículo de transporte do toner na "escova magnética" em c);

4o.)- uma série de diodos emissores de luz LEDs, em (d) acesos, descarregam todas as regiões. As regiões que receberam o toner permanecem ligeiramente carregadas, o suficiente para que este permaneça preso a sua superfície;

5o.)- no ponto "e" o papel é inserido/empurrado pelo cilindro alinhador (f), em "g" passa entre o corona de transferência e o cilindro fotocondutor. O corona de transferência é polarizado ao redor de 5000V, ficando assim o papel carregado negativamente, atraindo as partículas de toner positivas;

6o.)- logo após o corona de transferência, existe o corona de separação "h" polarizado com uma tensão AC em torno de 3600V. Este corona faz a dissipação das cargas, como num capacitor, eliminando a força de atração do papel com o cilindro, separando-os;

7o.)- o papel é arrastado pela bandeja succionadora "i" através de uma correia até o fusor. A sucção é feita por intermédio de um "fan",ou ventoinha. A sucção serve para prender o papel e arrastá-lo até o fusor, sem tocar no toner, ainda depositado sobre este.

8o.)- o toner é fundido no papel pela ação do calor e pressão devido aos rolos quentes do fusor "j". Estes rolos são mantidos a cerca de 190 C.

9o.)- uma lâmina plástica em "k" raspa o cilindro, tirando qualquer resíduo de toner remanescente neste, limpando-o.

10o.)- outra carreira de LEDs em "1" removem qualquer carga residual da superfície do cilindro. Esses LEDs, bem como aqueles da pré-transferência, somente ficam acesos durante o período de impressão, ficando durante o processo de espera apagados, visando não saturar a camada fotocondutora.

A impressão está feita e o papel com a imagem desejada é apresentado na saída dos rolos fusores. A seguir, serão discutidos detalhadamente cada subsistema envolvido.

II-4)- Alguns subsistemas de uma máquina eletrofotográfica

Os subsistemas podem ser divididos em quatro grupos principais, como segue:

a)- subsistema do processo eletrofotográfico;

b)- subsistema de movimentação do papel;

c)- subsistema ótico/laser;

d)- subsistemas auxiliares.

a)- os subsistemas do processo eletrofotográfico são:

a1)- sistema de polarização e limpeza do cilindro fotocondutor;

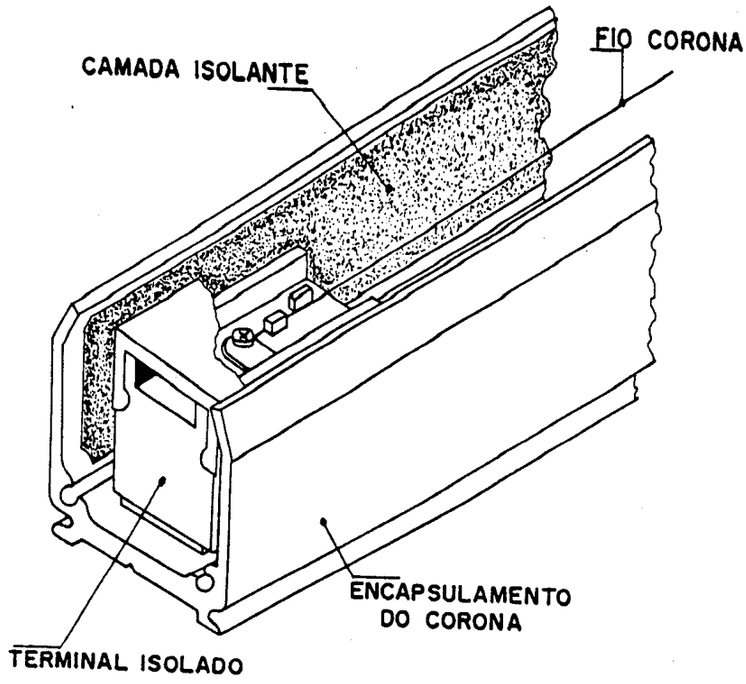
a2)- sistema revelador;

a3)- sistema fusor.

a1)- o sistema de polarização do cilindro fotocondutor é o responsável pela manutenção das cargas eletrostáticas ali necessárias (Schwiebert, 82). Na figura II-3 é mostrado um dispositivo típico. Normalmente, existem 3 dispositivos coronas. O primeiro é o responsável pela geração das cargas, que serão usadas no processo de atração das partículas de "toner", no módulo revelador. Geralmente é de 5KV.

O segundo é o de transferência, polarizado de forma que, quando o papel passe entre este e o cilindro recoberto de toner, as partículas sejam atraídas ao papel.

O terceiro é o de separação , normalmente polarizado com corrente alternada no valor de 4kV, visando eliminar as cargas no papel, de forma que este se separe do cilindro fotocondutor.



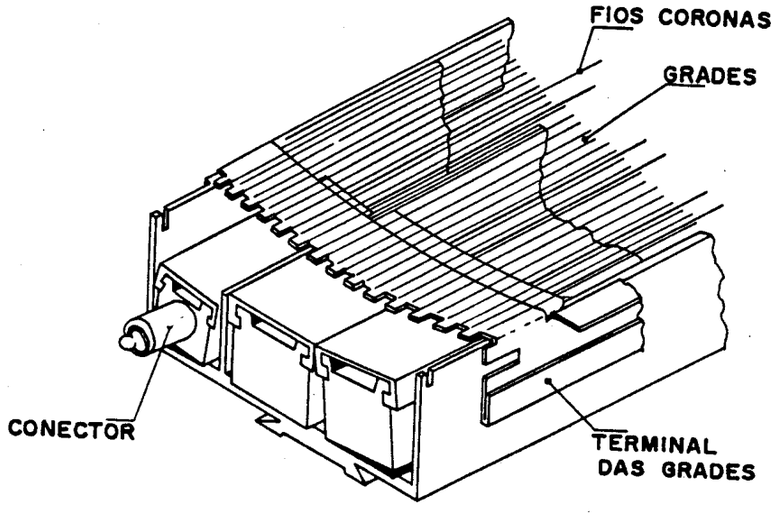


Fig.II-3)- Dispositivos corona típicos (Schwiebert,82)

Existem variantes do processo que usam mais de 3 coronas, uns para aumentar a eficiência do carregamento e outras para permitirem a inversão de imagem , com a utilização de fotocondutores recobertos por camadas isolantes (Shwiebert, 82).

Os coronas funcionam somente durante o processo de impressão, nos estados de espera ou aquecimento ficam desligados para não saturar a camada fotocondutora.

Existem duas carreiras de iluminação, por LEDs ou lâmpadas. A primeira é colocada, normalmente, antes do corona de transferência e é chamado de "Pre-transfer". Visa a diminuir as cargas de "back-ground", ou de fundo, residuais no cilindro e para facilitar o trabalho do corona de transferência. A outra carreira, chamada de "cleaning lamp", de LEDs ou Lampadas halógenas, é colocado antes do corona principal. Destina-se a eliminar qualquer carga residual, equalizando a superfície do cilindro fotocondutor. Ambas as carreiras somente funcionam durante o processo de impressão, ficando desligados no estado de espera.

O cilindro fotocondutor é acionado por um motor, sendo seu movimento disparado pelo início da impressão e sincronizado com os sinais de varredura do laser e com os rolos alinhadores/posicionadores de papel. O mesmo motor aciona uma rosca de Arquimedes, responsável pelo arrasto do toner residual até um reservatório externo. Este toner residual é devido ao fato de que nem todas as partículas de toner são transferidas do cilindro para o papel, sendo portanto, necessária sua remoção. A remoção é feita por meios mecânicos, normalmente por intermédio de "raspagem" executada por duas lâminas plásticas.

A lâmina inferior (mais maleável) fica em permanente contato com a superfície do cilindro e concordando com o sentido de giro do mesmo. Tem a função de desaglomerar o toner residual, permitindo, assim, um melhor desempenho da lâmina seguinte. A lâmina superior , chamada comumente de "Cleaning Blade", é colocado imediatamente antes do "Cleaning Lamp", constituída de material plástico, especialmente desenhada a raspar sem danificar a superfície do cilindro. Somente atua sobre este quando acionado por um solenóide . Este solenóide é ligado somente durante os períodos de escrita, visando evitar deformações permanentes na lâmina, que perderia sua eficiência. Esta lâmina atua de forma contrária ao sentido do movimento do cilindro, sofrendo, assim, grande desgaste e deformação, (Harpavat, 79).

Em alguns sistemas, principalmente os de alta velocidade, as lâminas são substituídas por escovas que giram em alta rotação, em sentido contrário ao do cilindro e junto com um forte aspirador dirigem o toner residual para um reservatório especial, as vezes para reutilização (Nebenzahl, 80), Fig II-4.

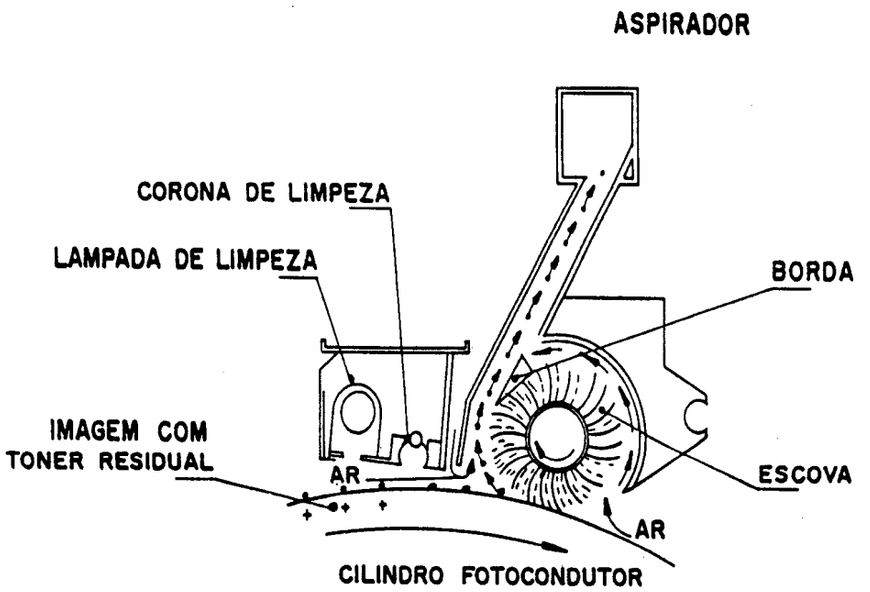


Fig.II-4)- Sistema de limpeza para o IBM 3800 (Nebenzahl,80)

a2)- sistema revelador

O sistema revelador ou "developer" é o responsável pela deposição do toner onde deverá ser formada a imagem que será impressa. Nas máquinas utilizando o processo de "xerografia" ou "copia seca", o "developer" é formado por um dispositivo chamado "escova magnética" (Schein, 75), (Takahashi, 82), (Benda, 81), (Nelson, 78).

Tal dispositivo é inteiramente "seco", por não utilizar "líquidos dispersantes" como nos processos "Nashua" (Tu, 75).

Neste dispositivo o toner é misturado com partículas magnéticas e a mistura é agitada por intermédio de fusos Devido a essa agitação e também pela polarização do dispositivo, através de uma derivação da alta tensão do corona de transferência, o toner adquire a carga eletrostática adequada para a atração pelas regiões ainda carregadas no cilindro fotocondutor. A agitação produz carga devido ao fenômeno de "triboeletricidade" entre os dois componentes, o toner e o "developer" magnético (Cammis, 82), (Bernardelli, 84).

A escova magnética é acionada por motores com velocidade constante. A "escova" é composta por uma série de imãs que giram dentro de um tubo de alumínio corrugado. Os imãs giram e arrastam as partículas magnéticas, o toner é então arrastado devido a atração eletrostática com as partículas magnéticas até a proximidade do cilindro fotocondutor (Iimura, 83). Nesse ponto, somente o toner é arrastado pela superfície carregada e o "developer" magnético permanece para ser reutilizado, fig II-5.

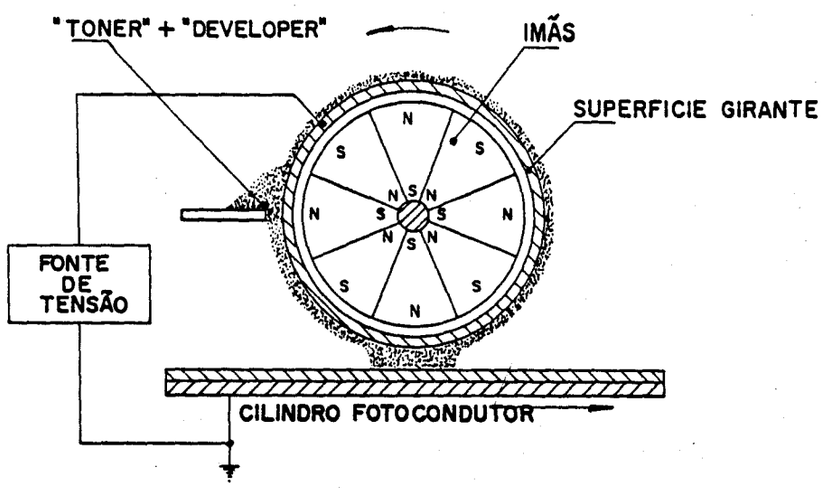


Fig.II-5)- Diagrama esquemático de uma escova magnética (Nelson,78)

Como se vê, é um mecanismo sofisticado, que exige um balanceamento preciso entre as cargas eletrostáticas. Geralmente, se usam dispositivos automáticos que compensem qualquer variação, mantendo as cargas no nível desejado, bem como a quantidade do toner que é misturado ao "developer". Tais dispositivos, chamados de "auto-toner", são densímetros de reflexão. A maioria consta de um LED infravermelho que ilumina um disco metálico não magnético polido, este em permanente contato com a mistura toner-developer (Lehmbech,79). A luz refletiva é captada por um fototransistor. De acordo com a quantidade de toner aderido neste disco de prova, o fototransistor recebe mais ou menos intensidade de luz. De acordo com este sinal, o "auto-toner" controla a relação da mistura toner-developer informando ao MCS a necessidade de se suprir ou não mais toner à mistura , através de um motor que arrasta este do reservatório de alimentação (Bustamante, 84), Fig. II-6.

Em outros sistemas, ao invés de se monitorar por um cilindro metálico , se monitora diretamente o "nível de preto" numa marca impressa na margem do papel sendo imprimido (Graf, 81), (Juve,82) Fig. II-7.

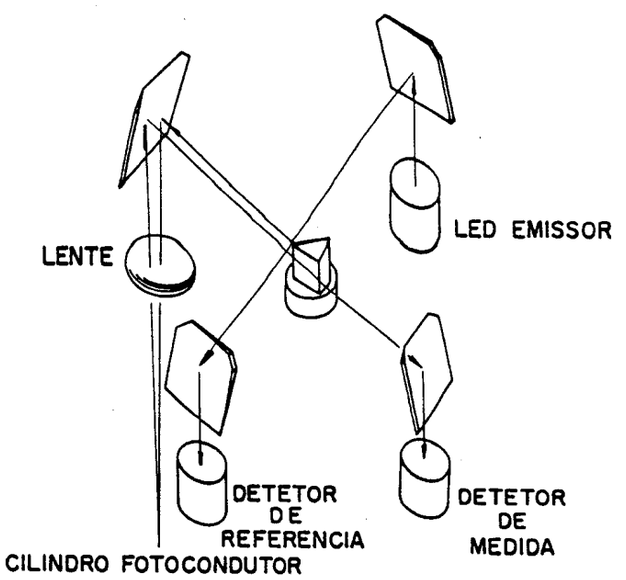


Fig.II-6)- Esquema geral de um "Auto-toner" (Sure,82)

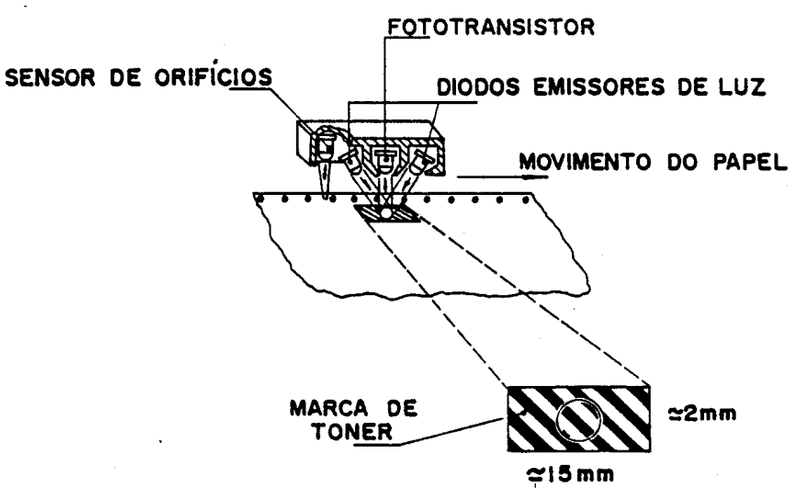


Fig.II-7)- "Auto-toner" da Siemens (Graf,81)

a3)- sistema fusor

O sistema fusor é o responsável pela fixação do toner no papel. Existem vários tipos, dependendo da velocidade desejada, tipo de toner, de papel (Prime, 83) . O mais utilizado nas máquinas de pequeno porte consta de dois rolos, onde passa entre eles o papel, sob pressão. O inferior é recoberto com silicone, não tomando contato com o toner e tem função tratora. O superior é recoberto com teflon, ou óleo de silicone, trabalhando em uma temperatura de 190 C. A temperatura é mantida através de uma resistência, ou lâmpada halógena, colocada no interior do rolo superior (Minor, 82), (Newkirke , 83), (Brooms, 78), Fig.II-8.

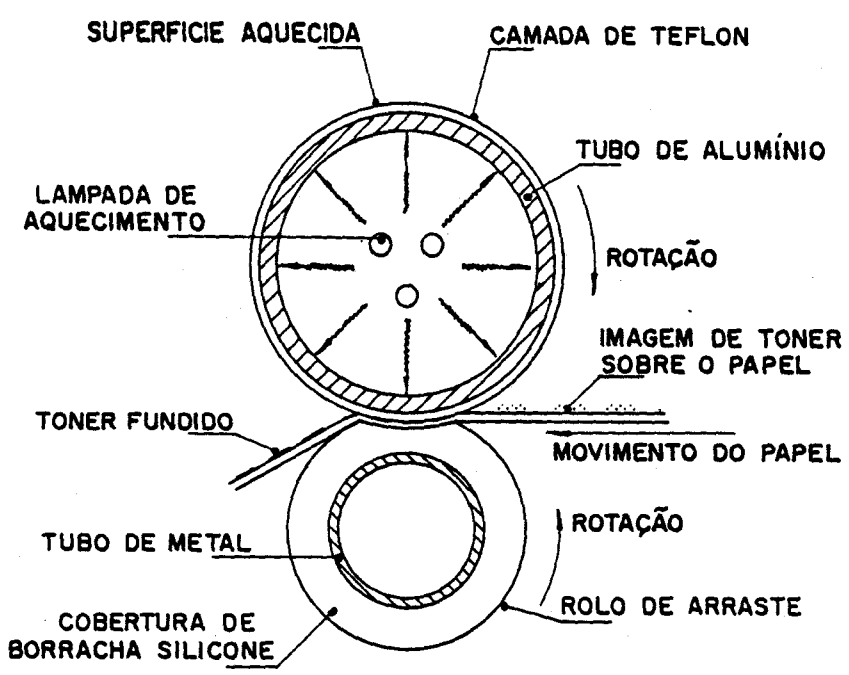


Fig.II-8)- Sistema Fusor IBM 3800 (Brooms,78)

Devido às temperaturas ali existentes, vários dispositivos de segurança normalmente são utilizados. Existem dois sensores térmicos. O primeiro envia sinais ao circuito monitorador da temperatura de trabalho, ligando e desligando a lâmpada de acordo com a necessidade. A lâmpada só é ligada se os rolos estiverem em movimento, visando homogenização. O outro sensor é um fusível térmico, que se rompe quando a temperatura se eleva demasiadamente. E colocado visando prever possíveis panes do circuito de controle, evitando, assim, que algumas partes entrem em combustão com a elevada temperatura. Este fusível corta a alimentação da lâmpada.

Os rolos fusores são acionados por motores de velocidade constante, normalmente "brushless"(sem escovas), esse mesmo motor é que aciona a correia da bandeja succionadora e os roletes de saída de papel.

Tal bandeja visa a arrastar o papel, desde o corona de separação até os cilindros fusores. A sucção é produzida por um "fan", ou ventoinha.

A função desta bandeja é a de compensar prováveis diferenças de velocidade no módulo fusor, o que poderia causar "jams" , ou atolamentos, no papel saindo do cilindro fotocondutor.

Na saída do módulo fusor existe um sensor mecânico, destinado a verificar a saída do papel. Caso este sensor não esteja acionado em determinado tempo, após o início do processo, o MCS interpreta que o papel ficou preso no caminho, interrompendo o processo.

Existem outros tipos de fusores, destinados a impressoras de alta velocidade. Funcionam baseados na exposição a radiação de lâmpadas de alta potência, exigindo sofisticado sistema de realimentação velocidade/temperatura, bem como de segurança (Archibald,82), (Wilson,79), (Baumann,84).

b)- o subsistema de movimentação do papel

Devido ao fato das impressoras a laser serem muito rápidas, o problema de movimentação e posicionamento do papel assume dimensões consideráveis no projeto da máquina, afetando a qualidade desejada de impressão , bem como a confiabilidade do sistema e seus custos (Svendsen, 78).

Os papéis podem ser tipo "formulário" contínuo, ou mesmo folhas A4 soltas. A escolha do tipo de papel induz a utilização de mecanismos e sistemas distintos de movimentação e controle (Borch, 84), pois são necessários a utilização de sofisticados sistemas de compensação de velocidade (Nakai, 82).

Nas máquinas de pequeno porte, usando folhas soltas, os sistemas prevêem, normalmente, a alimentação de papel por diversas vias, através de bandejas ou alimentadores manuais. Nas bandejas, existem sensores mecânicos que detectam a codificação presente neste, indicativa do tipo de papel que as mesmas contém, informando o MCS o tamanho máximo da página a ser impressa, caso a bandeja seja selecionada.

Também nessas bandejas existe um sensor que verifica se a bandeja está vazia ou não.

Recebido o sinal de início de impressão, o MCS aciona o motor-trator, que impulsionam os rolos que retiram o papel da bandeja. Muitas vezes, o motor girando em um sentido retira o papel de uma bandeja , ao se girar em sentido contrário retira o do outro. Isto é possível devido a utilização de um conjunto de engrenagens e catracas que somente transmitem movimentos num determinado sentido para os rolos de arraste. O uso de catracas é interessante por permitir que movimentos independentes sejam efetuados pelo mesmo motor, economizando motores, "drivers", etc.

O motor-trator impulsiona o papel até próximo do cilindro fotocondutor, onde existe um rolo chamado de "rolo alinhador". Neste local, um sensor ótico avisa o Controle Central, MCS, o posicionamento preciso da borda do papel. Neste ponto o motor fica esperando o sinal de início da escrita. Recebido o sinal de início, o motor dos rolos alinhadores passa a funcionar empurrando o papel em sincronismo com o cilindro fotocondutor e o laser. A velocidade do motor alinhador deve ser controlada, de forma a não haver deslocamento relativo entre a superfície do papel e a superfície do cilindro fotocondutor, para que não hajam distorções e borrões na impressão.

Devido a esses movimentos serem sincronizados, o uso de motores de passo é comum.

Em todo o percurso, o MCS espera o acionamento de sucessivos sensores mecânicos, indicando a presença do papel no percurso. Esses sensores devem ser acionados em determinado intervalo de tempo correspondentemente ao estágio em que o sensor se encontra, contado a partir do início do processo de impressão. Caso não haja acionamento no determinado instante, o controle central interpreta que o papel ficou parado no caminho, interrompendo o processo.

c)- subsistemas ótico/laser

O sistema ótico é o responsável pela geração da imagem no cilindro fotocondutor.

O cilindro fotocondutor, próprismente dito, nem sempre é um cilindro, sendo as vezes uma cinta que gira num percurso tracionado por roloc . A vantagem do uso da cinta é que permite uma melhor distribuição dos subsistemas do processo. A desvantagem é a vida mais curta, devido às deformações e possibilidade de vibração.

O materiais fotocondutores utilizados, dependem do comprimento de onda do laser e vice-versa. São utilizados, comumente, o selênio, o sulfeto de cádmio (Karan,82), fotocondutores orgânicos (Schaffert,71) e mais recentemente o silício amorfo .Os mais baratos são os orgânicos, porém possuem pouca vida útil em oposição ao silício amorfo, que devido a uma camada de proteção de nitreto de silício, possue vida teórica que se diz ilimitada (Nakayama, 82).

Os mais comuns são os de selênio. A camada é relativamente espessa (0,5 mm), depositada sobre alumínio. Porém, somente uma camada finíssima (60 um), constando de selênio dopado de telúrio, é realmente fotocondutora, daí sua fragilidade (Cheung, 82).

A sensibilidade fotocondutora do cilindro de selênio puro é maior ao redor de 560nm, neste caso, conforme o laser utilizado, necessita-se de potência diferente, a medida que o seu comprimento de onda se afasta deste pico.

Por exemplo, para se obter a mesma "descarga fotocondutora" com um laser de HeNe (632nm) de 1mW, um diodo laser infravermelho (780nm) necessitaria utilizar 5mW.

Com o advento dos diodos laser, com qualidades de economia de espaço e facilidade de modulação, houve um grande desenvolvimento dos fotocondutores orgânicos, trabalhando na faixa do infravermelho (Anon, 86), (Schaffert, 78), (Grammatica, 81).

A potência do laser influi na qualidade de imagem e velocidade de escrita. Quanto maior a potência, maior é a velocidade permissível para um mesmo material fotocondutor.

O laser deve ser modulado, para junto com a varredura, criar a estrutura "raster" dos pontos no cilindro. Devido as altas resoluções encontradas, normalmente de 160 até 2000 pontos por polegada, exige-se altas velocidades de modulação, variando de 1 a 20MHz. Para isso, são usados dispositivos moduladores de luz, tais como as células de Bragg (acústico-ótico) ou de Kerr (eletro-ótico). Os moduladores acústico-ótico são mais simples e baratos e são os mais utilizados nos sistemas de grande porte (Meye, 77).

Nos diodos lasers, a modulação é feita diretamente na alimentação deste, simplificando o sistema.

A varredura do laser pode ser feita tanto por galvanômetros, espelhos girantes, ou mesmo defletores acústico-óticos. Os mais utilizados são os espelhos poligonais, devido a boa relação custo/benefício (Fleischer, 77), (Grant, 79), Fig II-9.

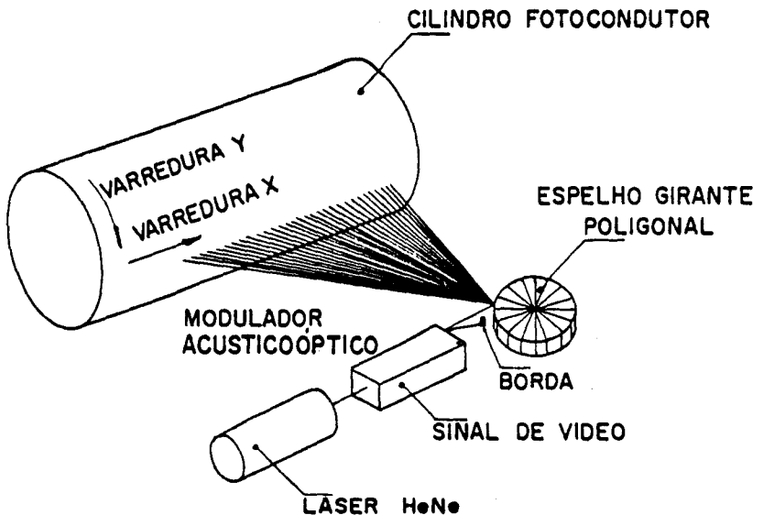


Fig.II-9)- Esquema básico de varredura do laser (Schwiebert,82)

O espelho poligonal é acionado por motores elétricos "brushless" (sem escovas), ou sincronos, para permitir um controle acurado de sua velocidade. Os mancais, normalmente, são a ar, pois as rotações são elevadas, da ordem de 20.000 rpm, permitindo frequências de varredura na faixa de 1600 a 5000 por segundo.

O uso de espelho poligonal é uma forma relativamente simples de se fazer tal varredura, mas obriga o sistema a ser tolerante com os desvios de fabricação. Normalmente são colocadas lentes e conjuntos óticos corretores de forma a minimizar os erros de face a face do espelho poligonal (Rabedeau, 78), (Fleischer, 73).

Outro problema é que a velocidade da varredura do feixe laser sobre o cilindro não é constante, bem como a distância relativa entre o laser e a superfície (Bergman, 76). O "spot", ou ponto focal, do laser tem necessidade de ser pequeno o suficiente para discernir um ponto impresso, ou seja 1/300 pol. nas impressoras lasers comuns. O telescópio colimador necessita ser colocado de forma a ter uma profundidade de foco aceitável, para que o tal "spot" não tenha muita variação. Quanto a velocidade não ser constante, torna-se necessário se utilizar de taxas de clock de modulação do laser variável durante o percurso, de forma a não distorcer a imagem. Vários esquemas são propostos, utilizando-se circuitos complexos (Lewis, 82), (Castro, 84), (Dattilo, 74), (Nikami, 82), (Shimado, 88).

A solução mais comum em produção de larga escala é a adoção de um conjunto de lentes, mais comumentes chamadas de F-teta. Este conjunto, quando bem desenhado, corrige a curva da velocidade, mantendo-a quase constante sobre toda a trajetória, permitindo "clock-fixo" (Takeoka, 79), (Shibuya, 83), (Hayashida, 83), (Maeda, 84), Fig. II-10.

LENTE F-TETA (Fθ)

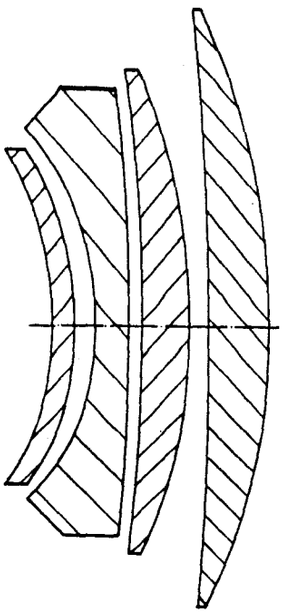


Fig.II-10)- Lente F-θ típica (Maeda,84)

Quando se utiliza de diodos lasers, a situação se complica um pouco mais, devido as condições do feixe emitida por este dispositivo. Nestes casos, é obrigatório o uso de um outro conjunto de lentes "corretoras" do feixe (Yamakawa, 89).

Para a sincronização do sistema são utilizados, também, vários esquemas. O mais comum é colocar no início da trajetória horizontal do feixe sobre o cilindro um fotodiodo rápido. Quando a varredura do feixe passa sobre o fotodiodo, este envia um sinal ao MCS e DCS permitindo o início da modulação do laser. Observa-se que tal sistema deve ser muito preciso, pois os erros não podem ser maiores que metade do menor ponto (Abe, 79).

O laser é modulado pelo sinal de "video" proveniente do DCS. Normalmente o feixe laser é desligado quando se deseja um ponto preto na impressão final.

Dessa forma, como as páginas impressas são na maioria "brancas" com pouca área "pretas", devido as letras, o laser fica a maioria do tempo ligado.

A adoção desse método permite a utilização dos cilindros e sistemas normalmente utilizados em máquinas copiadoras comuns, porém requerem maior precisão na montagem ótica, pois o feixe deve sobrepor-se, parcialmente, a cada varredura adjacente, sob a pena de aparecerem raias escuras no papel. Através de um artifício de reversão de carga e a utilização dos cilindros fotocondutores com uma camada isolante , as impressoras de pequeno porte usam o processo inverso , "acendem" o laser quando se deseja um ponto "preto". Isso torna o sistema mais tolerante aos erros de fabricação, pois as partículas de toner "encobrem" os erros. Conjuntos de lentes são desenhados para minimizar o efeito (Yamakawa,89).

d)- subsistemas auxiliares

Os subsistemas auxiliares são aqueles que efetuam tarefas periféricas no processo. São o Display, que informa ao usuário o funcionamento e configuração da máquina, o teclado, onde se aceita comandos de "on line", reset, escolha de bandejas, etc, os sistemas de dehumidifição para evitar o acúmulo de humidade, as lâmpada ou LEDs sinalizadores, bips, etc.

II-5)- O controlador central

O controlador central , ou "Machine Control System", é o responsável pela monitoração e controle de todo o processo físico de impressão (Crumly, 82), (Findley, 78),(Langley,82).

O MCS necessita:

a)- controlar o processo eletrofotográfico, mantendo-o dentro de parâmetros aceitáveis;

b)- controlar a movimentação do papel;

c)- verificar o funcionamento de cada sensor /atuador;

d)-notificar, durante o funcionamento, as condições reinantes na máquina;

e)- enviar, quando necessário, códigos de "status" para o "Controlador de escrita" e eventualmente enviar/receber comandos especiais de configuração;

f)- verificar a existência de situações emergenciais, para evitar possíveis danos à máquina ou usuário.

Como pode ficar aparente pelas discussões anteriores, o volume de processamento para efetuar todas essas tarefas é muito grande, exigindo estratégias similares aos sistemas "Multi-tasking" (Barrera, 84), (Miller, 84).

Geralmente, o MCS utiliza um sistema em tempo real, multitarefa, para administrar a máquina. Existem uma infinidade de tarefas que são executadas em aparente paralelismo, priorizados conforme sua importância. Tais prioridades são flexíveis, dependendo em que ponto do processo de impressão se está. O sistema também compartilha recursos entre tarefas concorrentes.

Uma importante função realizada pelo MCS é o de diagnosticar a máquina, permitindo ao usuário obter informações detalhadas de qualquer mal função que ocorrer e permitir "situações de teste", para maiores esclarecimentos .

O MCS também possui um canal de comunicação com o DCS, para tentar informar sobre parâmetros de configuração, quanto a eventuais erros ou falhas.

Tentou-se padronizar tal canal, porém , até hoje, não se obteve consenso (Russel, 84).

O MCS, normalmente, é constituído de um microprocessador ou microcontroladores de 16 bits, auxiliados por uma série de canais de entrada/saída, trabalhando em alta velocidade e independentemente do DCS.

II-6)- O controlador de escrita ou "Data Control System" (DCS)

O controlador de escrita é o responsável pela geração, na memória, da imagem a ser impressa no papel.

Tais controladores são chamados de "Raster Image Processors", por serem especialmente destinados a gerar imagens de forma "Raster", como nos CRT (Russel, 84), (Barret, 87).

Existe uma grande variedade de RIPs, os mais simples agem de forma similar aos controladores de vídeo, usados no CRT. Neste caso, todo o "cálculo" da imagem a ser impressa é feita no computador mestre. Neste caso, o "Data Control System" é simplesmente um dispositivo que necessita "plotar" os dados recebidos do computador mestre, não passando de um "interface de vídeo". Nestes casos, como se usa o potencial do processamento do computador, ou sistema hospedeiro, sofisticados softwares podem ser implementados, inclusive com facilidades "Multi-usuário" e grande capacidade de armazenamento de "imagens" prontas. Tal estratégia é muito usada nas impressoras de grande porte (McMurtry, 84). Porém, para computadores menores, o tempo gasto no processamento de funções gráficas e de composição da página é muito grande , tornando tal estratégia custosa.

Outra desvantagem significatica, neste caso, verifica-se também no tempo de comunicação. Se o computador tivesse que transmitir uma página A4 , com uma resolução de 300 pontos por polegada , por uma interface serial RS232 a 9600 bits/s, levaria cerca de 20 minutos. Tal fato obriga a transmissões por canais ultra rápidos, nem sempre disponíveis nos computadores e impressoras de pequeno porte.

A alternativa é de incorporar progressivamente a "inteligência necessária" nos controladores internos da impressora. Neste caso, o computador sómente necessita enviar comandos, instruindo o controlador das operações necessárias.

A impressora, neste caso, possui uma "linguagem" que pode ser usada por uma variedade de computadores diferentes, bastando conhecer tal linguagem.

O controlador que processa tal linguagem deve ser muito rápido e ter capacidade de manipular um volume muito grande de dados. Como já visto, uma página A4 a 300 dpi necessita 8.6 mega bits de memória para armazená-la.

Tal fato fez surgir alternativas que evitassem o uso de memórias tão vastas que, na época, não eram baratas. Geralmente, são usados três métodos (Douglas,87).

O primeiro deles é o "single line buffering", onde sómente uma linha é armazenada, sendo o Processador o responsável pelo envio de informções que serão impressas na linha corrente.

Tal esquema é interessante quando a página contém sómente texto ou gráficos de baixa resolução, pois permite que o processamento perceba quais "letras" estão na linha corrente.

O segundo processo utiliza-se de uma memória um pouco maior, de forma a ter-se um "multiple line buffering", ou "windowing".

Neste caso pode-se, por exemplo, ter uma memória suficiente para 1/4 da página A4. O DCS recebe as informações da página e calcula quais as partes que deverão ser acionadas dentro do "window" corrente,o primeiro quarto da página. Ao se terminar o "cálculo", o processo de impressão se inicia e o DCS move a "window" para o segundo quarto, assim por diante. A grande desvantagem desse método é que , em certas circunstâncias, a página deve ser "calculada" diversas vezes, no exemplo, 4 vezes.

O terceiro método é o de "Full Page Storage", onde existe memória suficiente para o armazenamento da imagem da página inteira. Ultimamente, é o método mais utilizado devido a redução do custo das memórias de alta densidade.

Este sem dúvida é o método mais versátil, pois não necessita de rotinas de "Windowing" ou "Pre-sorting", para verificar se os pontos estão na janela corrente. Em algumas impressoras existe memória suficiente para 2(duas) páginas ou mais, enquanto uma é impressa se calcula a segunda, (Ohr, 84).

Como na maioria das páginas o que existe é texto, alguns sistemas possuem a capacidade de efetuar certas operações via hardware, tais como: rotacionar, expandir e superpor caracteres (Gordon, 82). Como uma letra ou caracter muitas vezes é repetido na página, o próprio hardware já efetua tal "cópia" entre posições de memória (Barrera, 84).

A estrutura de um RIP destinado às impressoras laser de pequeno porte,é relativamente simples, como na Fig. II-11.

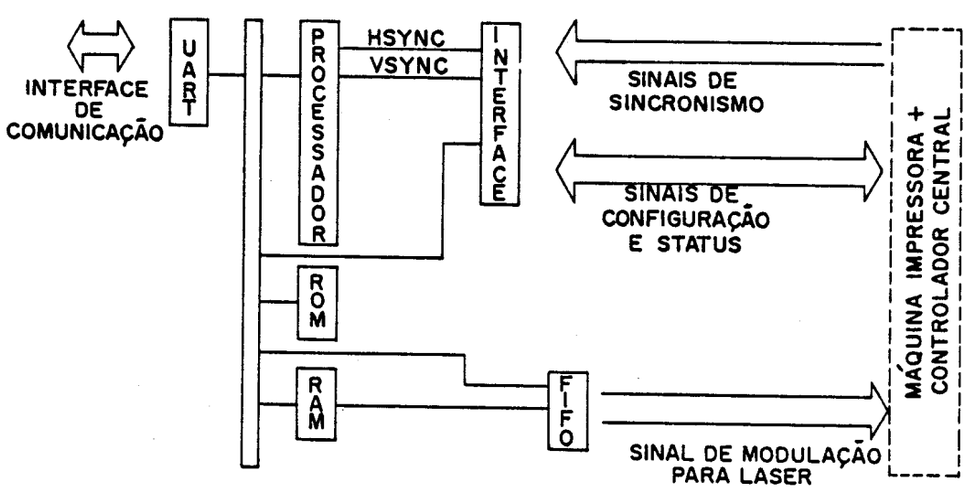


Fig.II-11)- Estrutura básica de um controlador de escrita

A interface de comunicação recebe os comandos que são processados, usando o programa residente no ROM. A página é montada no RAM que já possui um canal rápido de transmissão para a saída de vídeo, que irá modular o laser. Sinais de Sincronismo, configuração e "status" do mecanismo impressor são utilizados, tanto para gerar imagens na memória, quanto para garantir o posicionamento desta no papel.

A "inteligência" do RIP é representada pela "Page Description Language" (Fitzgerald, 82). Essa é a linguagem que a impressora entende e que monta a imagem da página. Os PDL, geralmente, contém comandos que descrevem os tipos de caracteres que serão usados, seu tamanho, orientação, etc. Deve possuir comandos que permitem o desenho de figuras normais, tais como: círculos, elipses, barras, linhas. Deve ser capaz de misturar texto com gráficos em toda a página. No próximo capítulo, será descrito o que é um PDL e suas características.

III- LINGUAGENS PDL

III-1)- Introdução

Neste capítulo serão apresentadas algumas linguagens PDL mais comuns e suas características descutidas através de exemplos.

Atualmente, com a existência de poderosas e flexíveis tecnologias de impressão, notadamente as eletrofográficas a laser, tem surgido a necessidade de transformar os antigos editores de texto em ferramentas mais poderosas. Os computadores de médio e pequeno porte, hoje, geram imagens complexas para uma variedade imensa de aplicações , tais como: CAD-CAM , "desktop publishing", etc. Um documento pode ser criado com estilo e características de impressão, somente encontrado em livros ou jornais impressos em máquinas tipográficas (Baker, 83), (Jurgen, 87), (Burkett, 85).

Estes sistemas requerem um método rápido e flexível de enviar tais imagens para a impressora.

Devido a essa necessidade, surgiram as "Page Description Languages", que virtualmente podem definir qualquer imagem enviada pelo computador , calcular e montar tal imagem de forma que seja impressa.

Tais PDL tornam o conjunto computador/impressora mais eficiente, pois ao invés de ser necessário enviar os milhões de bits de informação da imagem para a impressão a cada página, a linguagem de alto nível descreve os desenhos e caracteres via estrutura de dados e comandos, que o controlador da impressora converte diretamente no "bit-map", na memória (Bhatt, 86).

Na figura III-1 abaixo tem-se, aproximadamente, como deve uma PDL estar situada no conjunto Host/Printer.

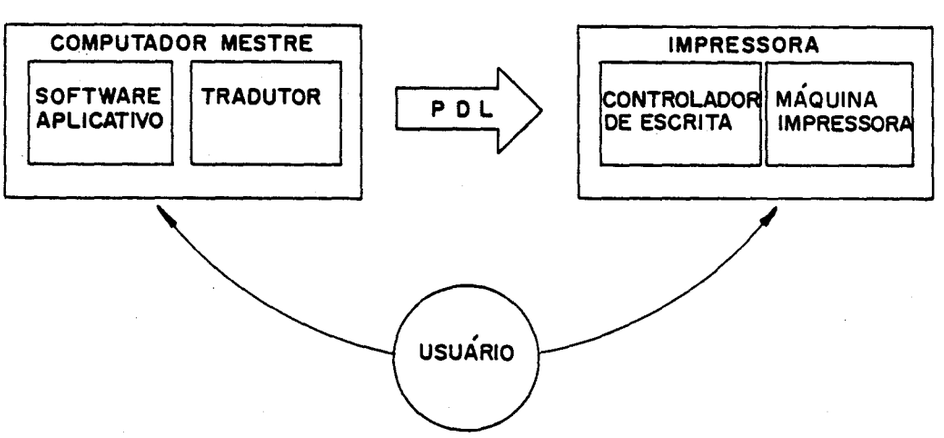


Fig.III-1)- Localização de um PDL num sistema

A linguagem reside parcialmente no computador mestre, parcialmente no aplicativo e também no Data Control System da impressora.

Dessa forma o aplicativo detém as suas funções específicas , não se preocupando com as características do mecanismo impressor. Quando o aplicativo envia os sinais correspondentes aos que se deseja imprimir, o Tradutor embutido, converte tais sinais nos comandos do PDL. O "Data Control System", na impressora entende tais comandos PDL e os retraduz, novamente, nas operações necessárias no mecanismo impressor.

Tal estratégia permite uma margem grande de portabilidade, permitindo que o software aplicativo se comunique com uma grande gama de impressoras a laser de média resolução (300 dpi). O mesmo conjunto de comandos, usados para gerar tal página nessa impressora pode criar páginas numa fotocompositora com resolução de 2000 dpi (dpi = pontos por polegada).

O PDL deve ser capaz, geralmente, de informar que tipo de caracteres serão usados e posicioná-los. Deve, também, ser capaz de rotacionar, ampliar ou reduzir desenhos e figuras básicas, tais como: círculos, elípses, linhas grossas ou finas. As PDL , devido a essa necessidade, tem um grande conjunto de comandos que poderiam tornar sua compreensão difícil e demorada. Para isso tais linguagens usam de "modelos gráficos", de forma a tornar o procedimento de "montagem" da página similar ao usado no processo real de fotocomposição tipográfico.

A seguir, serão discutidas algumas linguagens PDL e suas características básicas.

III-2)- Algumas linguagens PDL

Uma série de PDL foram criadas ao longo do desenvolvimento das impressoras laser e similares. Alguns tiveram maior sucesso que outros, porém, até o momento, não se estabeleceu nenhum padrão. Serão discutidos os mais comuns.

- Interpress do Xerox;

O Interpress foi criado em 1.982, para a utilização na sua impressora Xerox 1200. Permaneceu em uso exclusivo da companhia até 1.984 , quando algumas versões se tornaram públicas.

Tal linguagem é destinada a impressoras de alta velocidade, de forma que os comandos são abreviados. O texto é transmitido de forma compactada, sendo que a geração do documento é feito no Computador Mestre. Os caracteres são codificados em "bit map" de forma compriminda , onde sómente os pontos escuros são memorizados, economizando assim memória.

Os caracteres não podem ser expandidos, devendo o usuário adotar o "fonte" na forma original, ou escolher outro do tamanho desejado. Não são permitidos rotações de gráficos ou textos de forma arbitrária, sómente em múltiplos de 90 graus.

O modelo gráfico utilizado é o de "planos gráficos", ou seja, tentam simular os "Stencils" do processo "off-set".

Uma vantagem do Interpress é que, ao invés de se definir página por página, o usuário pode produzir os comandos para a impressão do documento inteiro, armazenando o tamanho da página, indices, caracteres usados, formatos padrões, etc.

O Interpress permite, também, a sua utilização em ambientes multi-usuários.

- O DDL da Imagem Corp.

O DDL da Imagem Corp. foi criado em 1.981 para sua impressora laser. Como, na época, era muito custoso ter-se em tese 1,5 Mbytes de memória para armazenar a imagem da página inteira, o modelo gráfico utilizado simplificou e diminuiu tal necessidade.

A técnica consistia em considerar a página como uma matriz de pequenas minipáginas. A resolução destas minipáginas era variável. Por exemplo, se em uma minipágina somente houvesse texto, o sistema armazenava a informação de codificação de caracteres que seriam usados, tais caracteres estavam na ROM da linguagem. Assim, tal parte da página gastaria pouquíssima memória. Se outra minipágina tivesse que usar gráficos de alta resolução, então tal minipágina seria armazenada na RAM.

Como a maioria das páginas são de texto, tal procedimento alivia de forma significativa a necessidade de memória. Tal esquema é utilizado pelo HP Laserjet, onde se especifica quais trechos são a 75 dpi (texto) ou a 300 dpi (gráficos).

A linguagem DDL possui comandos gráficos que desenvolvem figuras básicas, tais como : círculos, elipses, etc, bem como gráficos especiais, tais como : "pie charts", "bar graphs", etc.

A linguagem utiliza comandos altamente compactados, tais como sequências de " escapes ", para permitir rápidas comunicações com o computador, porém pode ser " traduzido " em comandos mneumônicos que permitam ao usuário fácil compreensão.

O DDL também é uma linguagem destinada a criações de documentos completos, não simplesmente páginas, e suporta sistema multi-usuários.

- O DCF da I.B.M.

O "Document Composition Facility" da IBM é o PDL criado para equipar a impressora de grande porte IBM 3800. Surgiu em 1974 e seu processamento é dividido entre o Computador Mestre e a impressora.

Na preparação do documento, no Mestre, o sistema automaticamente já configura a impressora, cria a tabela de "fonts", "Forms Overlays", etc. Em outros palavras, o documento é "pré-compilado".

Tal linguagem é fortemente dependente do hardware, para cada classe de comando existe um hardware dedicado e permitindo que o processamento final da página se faça no momento da impressão. Existe memória suficiente para o armazenamento de várias páginas na resolução limite de 175 dpi, permitindo a impressão de gráficos na página inteira.

Apesar de antiga, o DCF previa novos desenvolvimentos e ainda é considerada uma das mais poderosas linguagens para o sistemas de grande porte.

- O POSTSCRIPT da ADOBE SYSTEMS

O PostScript foi criado em 1985, por pesquisadores oriundos da Xerox e a primeira impressora a ser equipada foi o Apple LaserWriter.

O PostScript é uma linguagem interpretativa e procura ser independente da máquina utilizada. Em outras palavras, o usuário programa da mesma forma, tanto para uma impressora de 300 dpi quanto para uma fotocompositora a 2000 dpi.

A linguagem PostScript é uma linguagem de uso geral, muito parecida com a linguagem FORTH, adicionada com um grande número de comandos especialmente destinados a compor caracteres, perfis, imagens digitalizada em preto e branco, ou até mesmo em cores.

Os caracteres, em PostScript, são definidos de forma matemática, permitindo ser rotacionado e expandido em qualquer orientação. A vantagem é a flexibilidade e a qualidade obtida, já que o caracter nunca sai com os defeitos, normalmente encontrados ao se expandir uma definição em "bit map" . A grande desvantagem é o tempo de processamento.

O mesmo ocorre com figuras, desenhos, etc, que sempre são tratados de forma matemática. Por isso, conclui-se que o PostScript não destingue caracteres de desenho, sendo tratados da mesma forma, aumentando muito sua versatilidade.

A comunicação é feita para comandos mneumônicos em alto nível, sempre enviados em ASCII, sendo, portanto, altamente legíveis.

O PostScript é uma linguagem destinada a definição de páginas, porém podem ser criadas estruturas de comandos específicos que permitam o processamento do documento como um todo, em várias páginas, apesar de não armazená-las simultaneamente.

O modelo gráfico é o mesmo adotado pela Xerox, ou seja, utiliza os "Stencils" ou "planos" de imagem.

O modelo gráfico permite a criação de imagens sómente encontradas nos grandes sistemas de fotocomposição, por exemplo, pode-se usar um perfil de um caracter como se fosse uma máscara sobre uma foto ou outro desenho. Outro exemplo, os caracteres são definidos por equações que definem seu perfil, tais equações podem ser usadas para criar "trajetórias" para dirigir um "pincel" especial.

A flexibilidade do PostScript é muito grande, porém, devido ao grande volume de processamento necessário ,ele é muito lento.

III-3) Alguns exemplos usando o PostScript

Serão, aqui mostrados, alguns exemplos de utilização da linguagem PostScript. Não se pretende entrar nos detalhes da linguagem mas sim mostrar, em linhas gerais, as suas características e limitações (Adobe I, II, 85)

a) Imprimir a palavra "typografy"

/Times-Roman findfont	Seleciona o tipo de caracter, no caso o "Times Roman".
15 scalefont	Fator de escala do caracter que será usado, aqui ele leva o tamanho de 15 pontos (15/72 pol)
Set font	Sela o font escolhido com o tamanho
72 200 moveto	Move o "pointer" para a posição X=72 e Y=200 da página.
(Typography) show	Imprime na imagem da memória
Showpage	Transfere tal imagem para o papel.

Na figura III-2,abaixo, é mostrado o resultado. Salienta-se que os caracteres são definidos por equações, de forma que é possível se utilizar de "fatores de escala", sem correr o risco de que ocorram erros de discretização, ou perda de definição do caracter. É possível expandir , rotacionar, adicionar, etc.

typography

Fig.III-2)- Exemplo a (Adobe I,85)

b) Imprimir um quadrado

Newpath	Indica o início de uma nova trajetória
270 360 moveto	
0 72 rlineto	Desenha a linha, do ponto atual até X=0 Y=72 relativamente
72 0 rlineto	
0 72 rlineto	Idem, desenhando a linha
Closepath	Fecha a trajetória e cria a linha do último ponto até o início
5 setlinewidht	Define espessura da linha
stroke	"Risca"
showpage	Mostra a página impressa

Neste exemplo, observa-se o conceito de "trajetória" . Uma trajetória pode ser desde linha até curvas, ou mesmo "caracteres". Somente depois que a trajetória é definida, é que o "pincel" vai "riscar", usando tal trajetória .

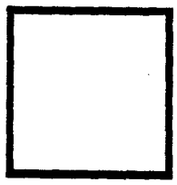


Fig.III-3)- Exemplo b (Adobe I,85)

c) Exemplo de utilização da área de mascaramento

/trianglepath	Cria uma subrotina chamada de "trianglepath"
{	Início
newpath 0 0 moveto 144 0 lineto 72 200 lineto Closepath	Cria a trajetória com a forma do triângulo
}	
def	Fim da subrotina "trianglepath"
/Vertical	Outra subrotina, que cria linhas verticais
{newpath 0 9144 { 0 moveto 0 216 rlineto } for shoke	Laço iterativo
}	
def	Fim da subrotina
/horizontals	Idem para linhas horizontais
{newpath 0 10 200 { 0 exch moveto 144 0 rlineto } for shoke	Laço iterativo
}	
def	Fim da subrotina
% begin Program	Início do programa principal
230 300 translate	Translada a origem do sistema de coordenadas.
trianglepath Clip	Define a trajetória "triangLepath" como a máscara
vertical	Desenha linhas verticais
horizontal	Desenha linhas horizontais
showpage	Imprime a página

Neste exemplo, somente o retículo que cai dentro do triângulo é que é desenhado, Fig III-4.

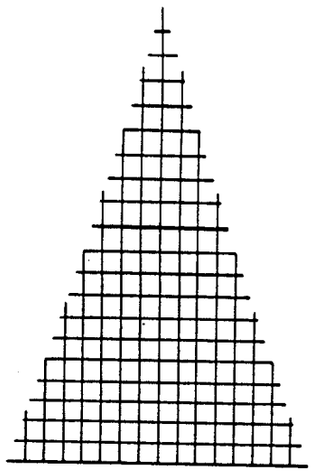


Fig.III-4)- Exemplo c (Adobe I,85)

Como observa-se pelo exemplo, o PostScript foi criado usando modelos gráficos, que permitiam as noções de "trajetórias" , "máscaras", "pincel".

Tais noções são importantes pois permitem ao usuário que compreenda as reais possibilidades do conjunto de operações disponíveis.

A adoção de um "modelo gráfico" é , portanto, de grande valia para que a linguagem atinja seus fins.

IV)- MODELO DA LINGUAGEM PROPOSTA

IV-1) - Introdução

Nos capítulos anteriores foi discutido o funcionamento aproximado das impressoras laser, algumas características internas e algumas características de linguagens PDL comuns.

O autor deseja, agora, iniciar a apresentação de sua proposta.

Tal linguagem visará :

a)- Ser de fácil assimilação, usando um modelo gráfico aprazível e similar ao processo real de composição tipográfica;

b)- Ser rápido, explorando as características de processadores gráficos;

c)- Ser expansível, para abrigar novas funções personalizadas;

d)- Ser independente do hardware impressor utilizado, para permitir certa transportabilidade;

e)- Ser, quando necessário, de fácil comunicação, utilizando comandos mneumônicos próximos ao significado real e físico.

Para o melhor entendimento das possibilidades do modelo gráfico, será iniciada a discussão pelas características do processador gráfico que será adotado.

IV-2)- Recursos gráficos disponíveis no TMS34010

O TMS34010 é um microprocessador, especificamente desenhado para operar em ambientes gráficos. Possui um conjunto de instruções de uso geral e outro específico para as aplicações gráficas (Texas, 86).

Foi o primeiro processador a combinar as duas características acima e sua aplicação básica é a de formar terminais gráficos "inteligentes", ou mesmo "placas gráficas" autônomas para micros pessoais, visando uma velocidade maior de processamento de programas gráficos.

Suas potencialidades são grandes, mas se situam numa faixa intermediária entre os processadores de uso geral e os "Raster Image Processors", dedicados às grandes estações gráficas.

Sua utilização, como controlador de impressoras laser, é sugerido pelo catálogo do fabricante (Guttag,88).

As funções gráficas embutidas permitem :

a)- manipulação de blocos bi-dimensionais de pixels, os "pixel block transfers" ou PIXBLT ;

b)- "Raster Operations" , que permitem executar operações booleana e aritméticas nos PIXBLT ;

c)- definição de janelas "windows", com detecção de violação, intersecção e movimentação;

d)- definição de máscaras, onde as operações PIXBTL são inibidas ou modificadas, permitindo efeitos de transparência e "clipping";

e)- rotinas dedicadas a traçagem de linhas, procedimentos incrementais permitindo a implementação de rotinas de geração de figuras básicas;

f)- formas de endereçamento programáveis, tais como X-Y ou linear, tamanho de pixel selecionável de 1 a 16 bits em potência de 2, sistemas de coordenadas absolutas ou relativas.

Os PIXBLT , imediatamente, sugerem a utilização de "telas", que podem ser transferidas ou compostas umas com as outras. Por exemplo, um caracter é definido em uma minitela, toda vez que tal caracter é chamado, é feito uma transparência da minitela para a posição que o caracter vai ficar.

Os PIXBTL permitem que se efetuem operações durante as transferências. Isso permite, por exemplo, a sobreposição de dois desenhos, a intersecção e "mistura", a "soma" e até mesmo "transparência". Tais operações chamadas de "Raster Op", são muito conhecidas nas literaturas especializadas e são parte obrigatória em qualquer dispositivo de processamento gráfico (Pountain, 87), (Guttag, 86). Tais operações sugerem, o conceito de "stencil", "máscara", "slide", que perfazem na pratica o que as operações "AND", "OR", fazem em "bit maps", (Salesin, 86), (Porter, 84), Fig. IV-1.

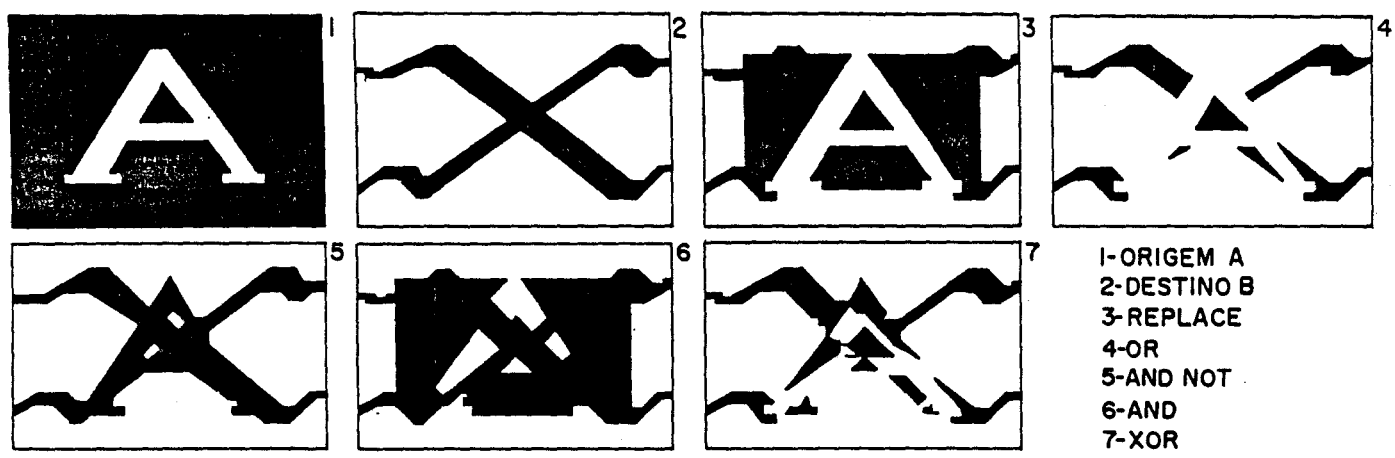


Fig.IV-1)- Algumas operações PIXBLT (Guttag,86)

O uso de coordenadas XY incrementais permite a caracterização das "trajetórias", que podem se repetir em qualquer ponto de origem no papel, como por exemplo, na definição de um logotipo ou mesmo de um caracter. Os caracteres seriam compostos pelas definições analíticas das "trajetórias" que o "pincel" deve efetuar.

IV-3)- Modelo Gráfico Adotado

O modelo escolhido foi o de simular o processo físico, relativo aos métodos de tipografia e fotocomposição. Dessa forma, o modelo, por si só, já determina a forma que perfis geométricos, tipos , desenhos e cores são combinados .

O modelo adotado corresponde, aproximadamente, ao processo de "silk-screen" . O papel recebe a tinta através de um "stencil" ou tela de "silk" . O mesmo papel pode receber sucessivas telas , combinando cores e formas. A "tela de silk" é desenhada com o auxílio de "pincel", "verniz", "negativos fotográficos" , podendo conter letras, fotos, desenhos, Fig. IV-2.

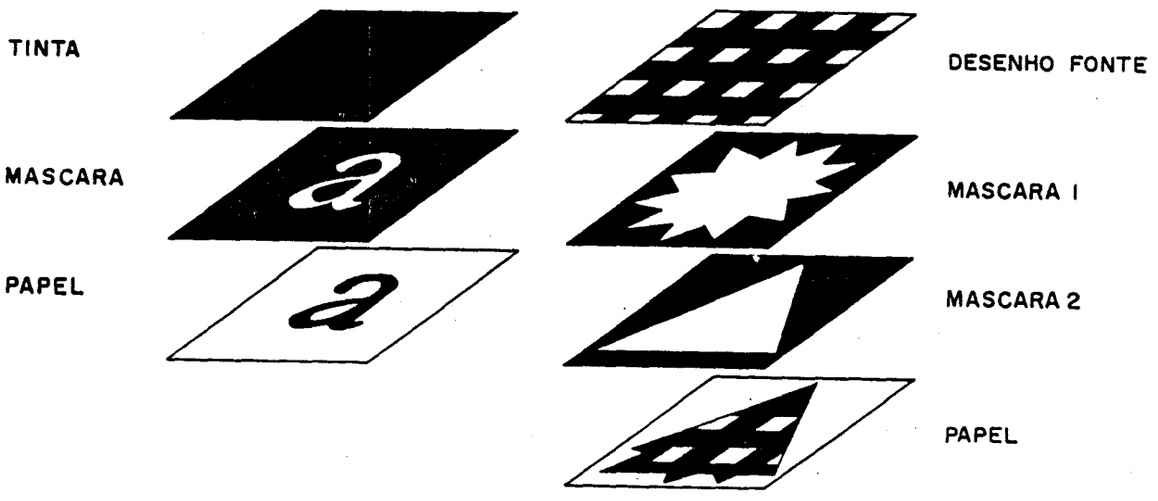


Fig.IV-2)- O modelo gráfico adotado (Warnock,82)

No modelo aqui sugerido , o usuário possue uma série de ferramentas para definir o "stencil" , outras para definir a "fonte de tinta" ou "source". Após definidas, o usuário pode combinar diversos "stencils" e "sources", sucessivamente. Os "stencils" podem ser utilizados como "formas" ou "máscaras". As "formas" constituiem-se de figuras, ou curvas, que representam os "buracos" por onde a "tinta" pode passar.Tais "formas" podem representar figuras ou letras, não havendo, portanto, distinção entre "texto" e "desenhos"

Este modelo simples tem uma facilidade de implementação muito grande, no TMS34010, devido as instruções de PIXBLT, permitindo inclusive a realização de operações, que somente seriam encontradas analogias físicas nos grandes sistemas OFFSET e de fotocomposição. Como por exemplo: os "sources" ou "tintas", que podem representar, desde simplesmente a tinta monocromática que iria passar pelo stencil , como pode também ser uma imagem bidimensional produzida por um processo anterior de "source-stencil" , similar ao que ocorre no processo de impressão "OFFSET" (Thoma, 81), (Warnock,82).

As operações booleanas e aritméticas, nos PIXBLT, permitem tratar o stencil muito mais que uma região de passa/não-passa. É possível somar , subtrair , saturar , sombrear , expandir , retacionar, como num processo fotográfico, (Porter, 84).

IV-4)- Algumas possibilidades usando o modelo gráfico adotado

A simplicidade do modelo não inibe sua capacidade de resolver muitos dos problemas comuns encontrados, na necessidade de se imprimir documentos. Alguns exemplos:

- Artigo científico contendo símbolos químicos:

O usuário deseja imprimir uma fórmula química , contendo, por exemplo uma grande cadeia polimérica. Infelizmente, a impressora não tem o símbolo desejado. O usuário, então, define uma "trajetória", usando os comandos disponíveis de reta , curva , pontilhado etc . Tal "trajetória" passa a existir e residir na impressora e futuros artigos poderão chamá-lo e utilizá-lo. O mesmo raciocinio se aplica a logotipo de firmas , margens personalizadas etc.

- "holerith" de pequena companhia:

O usuário necessita imprimir os "holeriths" do pagamento. Como, usualmente , o "holerith" tem uma série de campos fixos , tais como: o campo com o nome da firma , CGC , endereco , colunas explicativas, margens . Existem, também, o logotipo da firma, caracteres personalizados etc. O usuário cria então um "stencil", contendo todo o conjunto de "imagens", que não mudam de um "holerith" para outro e armazena com um nome. No momento da impressão , o nome do empregado e seu salário são colocados em outro "stencil" . Os dois "stencils" são conbinados mediante uma "ação gráfica" e no papel obteremos o "holerith completo".

- Cartaz promocional contendo foto;

O usuário necessita imprimir uma imagem/fotografia, que no momento foi adquirida por um scanner e reside no micro. Necessita imprimir tal foto juntamente com alguns comentários , porém, nem toda a foto é necessária , sómente uma região é de interesse.

Primeiramente, o usuário define como sendo o "source" a foto e define um "stencil" contendo uma "MASCARA" sob a região de interesse. Executa uma ação gráfica , utilizando o "stencil" como "clipping área" ou máscara. Tal imagem resultante passa a ser um outro "stencil", onde serão "somados" a imagem produzida pelos caracteres do comentário desejado.

V)- IMPLEMENTAÇÃO DA LINGUAGEM PROPOSTA - NÚCLEO BÁSICO

V.1)- Introdução

Neste capítulo será apresentada a estrutura básica da linguagem proposta.

Como visto no capítulo anterior, as principais PDL tem uma estrutura linguística que permite expansibilidade e modularidade.

O Postscript tem uma semelhança muito grande com a linguagem FORTH.

Foram estudadas as características dessas linguagens, seus detalhes de implementação , etc e baseado no conjunto de idéias e conceitos obtidos, criou-se um núcleo de forma a explorar ao máximo as possibilidades do processador escolhido, tanto quanto permitir a expansibilidade capaz de criar os comandos e estruturas necessárias para implantar o modelo gráfico adotado.

A linguagen proposta será, daqui em diante, chamada de "Miniscript".

Através do uso de uma série de exemplos serão introduzidos os conceitos e principais comandos do núcleo básico.

V.2)- Linguagens Interpretativas Encadeadas

A linguagem proposta Miniscript, como o FORTH, pertence a uma certa classe de linguagens chamadas "TIL" de "Threaded Interpretive Languages", que pode ser entendido como Linguagens Interpretativas Encadeadas (Loelinger,81). Tal classe agrupa um conjunto de linguagens que não se enquadram totalmente na classe de linguagens interpretativas, nem na das linguagens compilativas.

Uma linguagem compilativa recebe a linguagem fonte, que através de um programa tradutor, converte em códigos executáveis posteriormente (exemplo : Fortran, Pascal).

Uma linguagem puramente interpretativa recebe a linguagem fonte como entrada, para cada comando executa uma busca e encontrando a sua definição o executará diretamente.

A maioria dos interpretadores, porém, executam tal trabalho em duas fases. Na primeira fase, durante a entrada dos comandos ou edição, o interpretador traduz tais comandos para uma forma interna intermediária. Na segunda fase, durante a execução, é que as formas internas são interpretadas e executadas diretamente (exemplo: Basic).

Uma "TIL" produz uma forma interna intermediária já completamente analisada, ou seja, pronta para a execução.

Tal forma interna contintui-se de uma lista de endereços das funções utilizadas, definidas anteriormente. Esta lista é montada (ou "costurada" "threaded") durante a etapa de tradução, agindo de forma muito similar à ação de um compilador.

Por esta razão , as "TIL" e a linguagem "Miniscript" possuem dois estados fundamentais. O estado ou modo compilativo é onde se criam novas funções, baseadas nas definições e comandos já previamente existentes, organizadas numa lista de definições chamada dicionário. Tal nova função passa a fazer parte desse dicionário. No modo interpretativo o sistema procura a função desejada na lista "dicionário" e, se encontrada, é imediatamente executada sem que em seu interior haja qualquer nova busca ou tradução.

Para que se entenda melhor esses estados é necessário compreender a estrutura básica da linguagem, que é justamente este "dicionário".

Qualquer função, do sistema ou criada pelo usuário, faz parte e tem uma entrada neste dicionário. O dicionário nada mais é que uma lista encadeada de definições, sendo que cada definição tem um elo para a função definida imediatamente anterior. No corpo da definição estão as chamadas às funções que estavam no dicionário anteriormente.

Como exemplo, suponha-se que o usuário já criou algumas funções para uma aplicação específica, são elas:

```
ENCHE
ESFREGA
ENXUGA
```

Quando o usuário deseja executar a função ENCHE ele fornece ao sistema tal palavra. O sistema está no modo interpretativo e assim passa a procurar via dicionário tal função. Ao encontrar, pula para o corpo da definição de ENCHE e o trabalho é executado. A seguir o usuário envia o comando ESFREGA. Novamente, no modo interpretativo, a função é procurada e executada. O mesmo ocorre com ENXUGA.

Vamos supor que o usuário vai executar muitas vezes tal procedimento, de forma que ele decida criar um programa, chamado LAVA. Neste momento, o usuário entra então com:

```
: LAVA  ENCHE  ESFREGA  ENXUGA ;
```

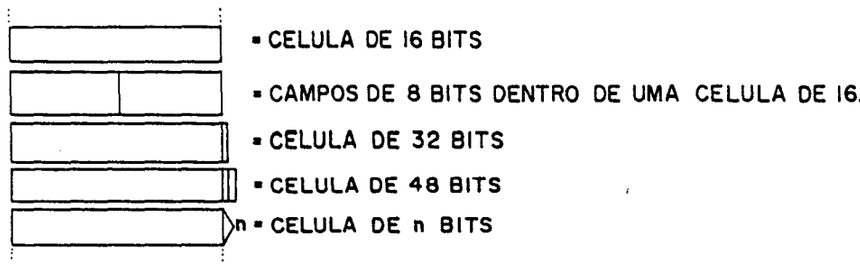
O comando ":" (Colon) é recebido pelo sistema e procurado pelo dicionário. No dicionário, sua execução coloca o sistema no modo compilativo e imediatamente cria uma nova função, cujo nome desejado está logo a seguir, "LAVA".

A partir daqui o sistema pega a próxima palavra, ENCHE, faz a busca no dicionário e quando o encontra ao invés de executa-lo, copia o seu endereço de execução no corpo de LAVA, que está sendo montado no dicionário. O mesmo ocorre com os comandos ESFREGA e ENXUGA. Ao se encontrar o comando ";" (semi-colon) o sistema entende que se chegou ao fim da função sendo criada e retorna ao modo interpretativo.

A partir daí, está disponível no dicionário a nova função, LAVA, e em seu corpo estão os endereços de execução de ENCHE, ESFREGA, ENXUGA.

Assim, quando o usuário envia o comando LAVA, no modo interpretativo, sómente uma busca é feita, sendo a velocidade em muito aumentada.

À seguir serão discutidas as estruturas internas adotadas no Miniscript. A seguinte conveção será adotada, na descrição dessas estruturas na memória:



Algumas estruturas manterão o nome original em inglês, visando manter o jargão já utilizado universalmente pelas linguagens "TIL".

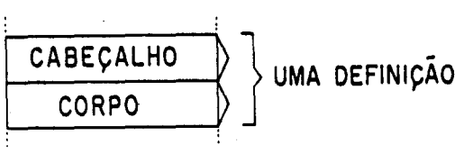
V.3)- O Dicionário

Como visto, qualquer operação ou função, do sistema ou do usuário, tem uma entrada no dicionário. Todos os aplicativos e virtualmente todo o sistema consiste de entradas ou chamadas ao dicionário. Tal dicionário compõe-se de uma lista encadeada de definições que vai crescendo na memória a medida da necessidade que novas funções criadas exigem.

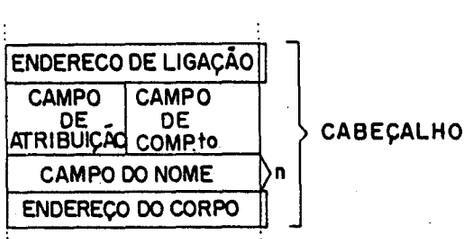
A forma com que uma definição é introduzida em tal dicionário varia de acordo com o objetivo da linguagem TiL desejada.

Existem uma série de diferentes formas de implementação , cada uma sintonizada a sua necessidade primária. Existem implementações onde a prioriade é a compactação (Loelinger,81), outras com prioridade na velocidade de busca (Ting,86).

No Miniscript a prioridade foi a velocidade de execução, de forma que uma entrada no dicionário tem a seguinte estrutura:



O cabeçalho (ou "header") pode ser subdividido da seguinte forma:



O campo do endereço de ligação (ou "link address field") é o endereço da próxima função existente no dicionário. É o elo da lista encadeada .

Este campo , de 32 bits , é utilizado na busca da função pelo dicionário .

O Campo de comprimento do nome (ou "number field"), de 8 bits, contém o número de caracteres utilizado na definição da função, ou seja, é o número de letras que o nome da função contém.

O Campo de atributos (ou "atribution field"), de 8 bits, contém alguns bits de controle que serão utilizados para diferenciar certas classes de funções. O primeiro bit é o "precedence" bit, ou bit de precedência que assinala se a função é do tipo "imediato". Funções imediatas são aquelas que são executadas mesmo no modo compilativo e serão discutidas adiante.

O Campo do nome (ou "name field") contém os caracteres utilizados no nome da função. Tal campo é sempre múltiplo par de 8 bits, ou seja, se o nome contiver 3 letras, são utilizados mesmo assim 32 bits.

O Campo de endereço (ou "body address field") contém o endereço da primeira instrução executável da função. Este campo é copiado pelo compilador no momento em que monta uma definição que usa esta função.

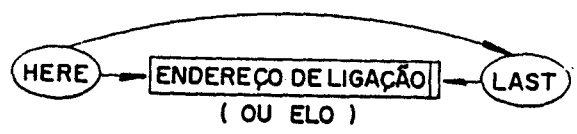
O Corpo da função (ou "body") contém as instruções executáveis que perfazem a função desejada.

Através do exemplo já visto da função LAVA, pode-se observar o que ocorre no dicionário quando tal função é criada.

: LAVA ENCHE ESFREGA ENXUGA ;

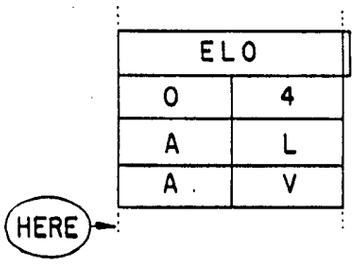
O sistema está no modo interpretativo e ao se encontrar a função ":" (Colon), entra no modo compilativo . Imediatamente, o sistema pega a próxima palavra ("LAVA") e conta o seu número de caracteres (4). No modo compilativo o sistema sabe que irá colocar uma nova função no dicionário. O sistema guarda duas informações importantes para a movimentação do dicionário: o endereço na memória da próxima posição livre para o dicionário, ,denominado HERE, e o endereço do cabecalho da última função criada no vocabulário sendo usado (veja discussão a seguir sobre os vocabulários), que será denominado aqui de LAST.

O ponteiro HERE está apontando para o campo de endereço de ligação da nova função e o sistema transfere o endereço contido em LAST.



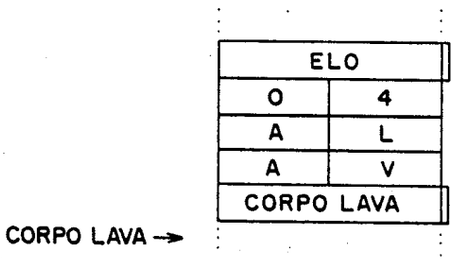
(Após tal operação, LAST contém o "HERE" pois LAVA passa a ser a ultima função criada)

A seguir, o sistema coloca o número de caracteres (4) e os atributos (o sistema sempre assume uma palavra não "imediata").



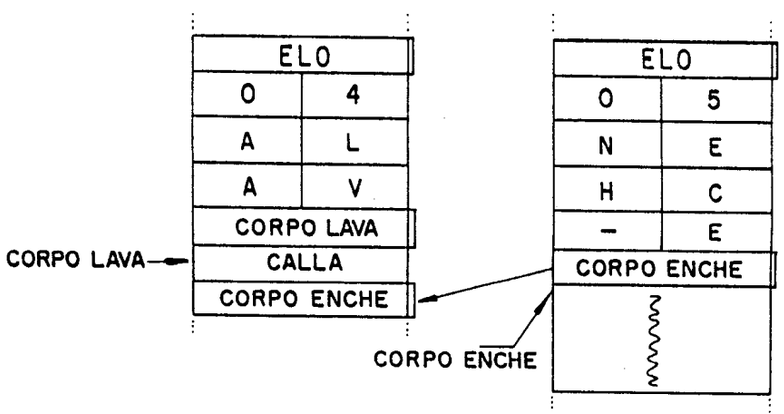
(Observe que HERE é incrementado automáticamente)

Após a última letra, o sistema coloca o endereço da primeira palavra executável do corpo que será, neste caso, o endereço HERE + 1 = CORPO LAVA



O cabeçalho estando pronto permite que o sistema passe, então, a procurar no mesmo dicionário as funções que comporão a definição , no caso a primeira é a função ENCHE . Ao encontrar ocorre o seguinte:

Como o sistema está no modo compilativo, ele insere no dicionário um "CALLA" (comando do TMS34010) e copia o "body address field" da função "ENCHE" no corpo da função criada:



O mesmo ocorre com as funções ESFREGA e ENXUGA.

Quando o sistema encontra a função ";" (semi colon) insere um "RETS" no dicionário (comando de retorno de subrotina do TMS34010) e recoloca o sistema no modo interpretativo.

Como se pode observar, no corpo da função LAVA só existem comandos executáveis.

Alguns comentários podem ser feitos a respeito desta estrutura.

Primeiramente, quando o sistema faz uma busca pelo dicionário, se utiliza de uma função interna chamada FIND.

Tal função pega o "endereço de ligação", contido no campo correspondente na primeira função deste dicionário , visando achar o "caminho" da lista encadeada.

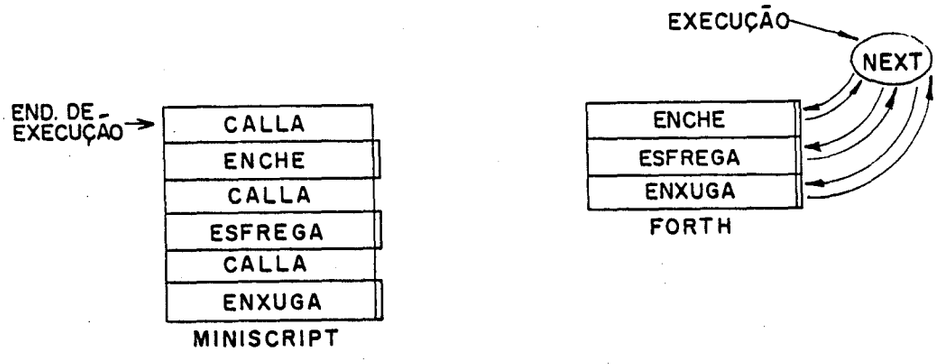
Ao se chegar em um cabeçalho, primeiramente é verificado se o número de caracteres contido no campo de comprimento é igual à função desejada. Se o número for igual, o sistema passa a comparar os caracteres, caso contrário, pula para o próximo "ELO" e se reinicia o teste. Comparando os caracteres ,se encontrar algum diferente também pula para o próximo "ELO".

Quando todas as letras são iguais o "endereço do corpo" fica disponível para ser, ou copiado (modo compilativo), ou executado através de um "CALLA" que o sistema faz indiretamente (modo interpretativo).

Tal forma de cabeçalho foi utilizado visando a otimização da rotina FIND no aspecto de velocidade e é similar ao utilizado por outras linguagens encadeadas.

O corpo da função é composto por um conjunto de "CALLAs", chamando as subrotinas apontadas por seus respectivos "endereços de corpo" . Tal estrutura foi adotada também por questões de velocidade e difere muito das estruturas de outras linguagens "TIL" como o FORTH.

No FORTH o encadeamento das funções é feito por uma rotina especialmente destinada, chamada NEXT. Assim, no corpo de uma definição FORTH não existem "CALLAS", sómente endereços, como na figura abaixo (Ting,86) (Loelinger,81) (Brodie,81). Tal esquema favorece a compactação, reduzindo em até cerca de 30%, a necessidade de memória para a compilação de uma nova definição, porém, como o encadeamento é feito por uma função, o processamento pode ficar até 3.5 vezes mais lento.



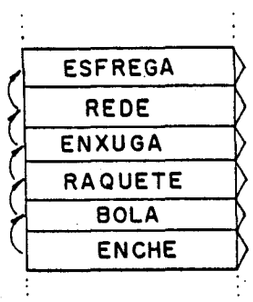
Como velocidade é um fator preponderante nos sistemas gráficos, a cadeia de "CALLAs" foi a forma utilizada.

O dicionário, como foi visto, é o núcleo da linguagem. Todas as funções são alocadas nele de forma encadeada, porém tais elos não necessitam ser feitos de forma a ligar sómente funções adjacentes fisicamente em tal dicionário. Tal encadeamento pode ser organizado de forma que certo grupo de funções formem um sub-dicionário ou vocabulário.

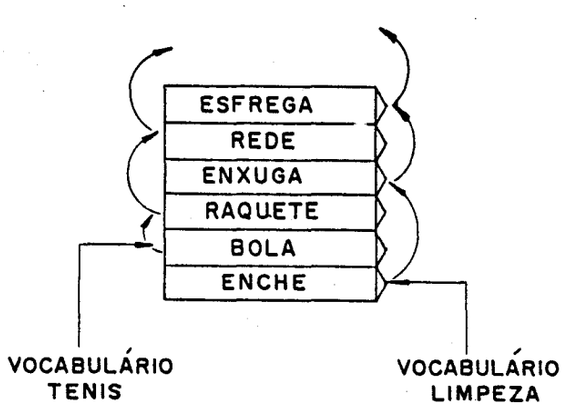
V.4)- Os Vocabulários

Vocabulários são sub-listas encadeadas de forma a agrupar palavras e funções afins. Sua principal finalidade é a de aumentar a rapidez do modo interpretativo e permitir que o processamento tenha um "contexto" preferencial, evitando assim o desperdício de processamento testando palavras que estejam fora deste "contexto".

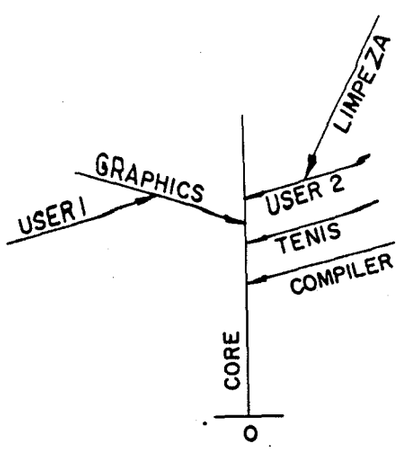
Se não houvesse os vocabulários o dicionário teria a seguinte aparência, tendo as funções sempre contiguidade física:



Porém, é possível que sejam agrupadas funções de um mesmo contexto em um encadeamento próprio, chamado vocabulário, dessa forma o dicionário teria a seguinte aparência:



Assim, o dicionário passa a conter uma série de encadeamentos que, apesar de compartilharem o mesmo espaço físico, não possuem encadeamento lógico. Os vocabulários permitem que o usuário organize sua aplicação , otimizando as ações do sistema. O usuário pode criar os vocabulários que quizer, seguindo uma forma hierárquica, como se fossem sub-listas que partem de uma lista maior ou principal, como em uma árvore (Vide fig. abaixo).



Originalmente o sistema possue 3 vocabulários, a saber:

CORE: contém as funções básicas do núcleo da linguagem onde se encontram as funções aritméticas básicas, operadores funções relacionais, utilitárias, funções de entrada/saída. E' a base do sistema.

COMPILER: contém as funções usadas no modo de compilação, as diretivas de controle de fluxo de processamento e as "defining words" que permitem a criação de novas estruturas de dados.

GRAPHICS: contém as funções dedicadas ao processamento gráfico, contendo a estrutura para suportar o modelo gráfico adotado.

Para o controle de tais vocabulários, o sistema mantém duas variáveis que contém os nomes dos vocabulários que estão sendo utilizados.

CONTEXT: contém o nome vocabulário onde as buscas (por FIND) serão feitas prioritáriamente.

CURRENT: contém o nome do vocabulário onde as novas definições serão colocadas.

Normalmente CURRENT = CONTEXT, porém, em algumas aplicações é interessante sua distinção.

Através do comando VOCABULARY o usuário pode criar o vocabulário desejado, tal vocabulário terá origem no vocabulário contido em CURRENT.

Por exemplo: O usuário vai iniciar uma tarefa gráfica. Para isso, deve entrar no vocabulário GRAPHICS, inserindo o nome:

GRAPHICS

Imediatamente: CONTEXT = CURRENT = GRAPHICS

Neste vocabulário, o usuário vai criar um novo vocabulário, por exemplo: USER1

Para isso:

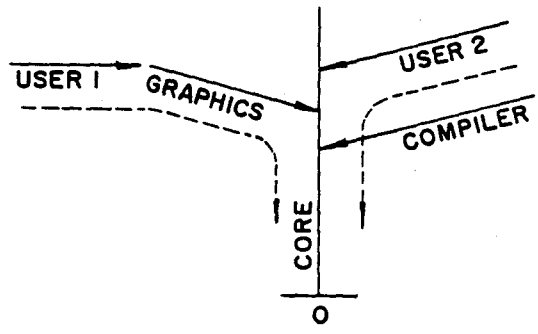
VOCABULARY USER1

Dessa forma o comando VOCABULARY criou, a partir de GRAPHICS, uma sub-lista chamada USER1. As próximas funções que forem definidas por ":" (Colon) serão adicionadas na sub-lista USER1, até que o usuário chame outro vocabulário.

Na verdade "VOCABULARY" cria uma variável que irá armazenar a última palavra definida nesta sub-lista.

Quando a função FIND inicia a busca, esta traz de CONTEXT o endereço da variável (vocabulário), que por sua vez contém o "endereço de ligação" da última palavra definida naquela sub-lista. A busca continua até que FIND encontre a origem da sub-lista, que por sua vez faz parte de outra, e assim por diante, até encontrar a lista CORE, que é a raiz do sistema. O fim de uma lista ocorre quando se encontra um zero no "endereço de ligação".

Na figura abaixo é mostrado o caminho que FIND faz quando CONTEXT = USER1 e quando, por exemplo, CONTEXT = USER2



O vocabulário COMPILER só é acessado durante o modo de compilação, independentemente dos vocabulários ativos no momento. A rotina FIND automáticamente passa pelo vocabulário COMPILER se o sistema estiver no modo de compilação.

Como vimos, o dicionário é organizado em vocabulários, sendo a forma adotada de armazenar e acessar as funções que compõe a aplicação desejada. Porém, não é o dicionário o responsável pela troca de informações entre as funções, ou seja, os dados que serão processados. Para isso, existe a segunda estrutura básica , fundamental nas linguagens "TIL" , que são chamados de pilhas.

V.5)- As Pilhas

As pilhas , ou "stacks", são as estruturas de dados que permitem a troca de informações entre os elementos de um programa ou funções. Nos TILs e no Miniscript, sempre os dados necessários a uma função provêem , direta ou indiretamente pelas pilhas.

Normalmente, existem duas pilhas principais:

- a "PARAMETER STACK", ou pilha de parâmetros é a utilizada para armazenar temporariamente os elementos de uma operação. Tal pilha participa de qualquer operação aritmética, de transferência ou manipulação e normalmente é conhecida simplesmente por "PILHA". É a estrutura que permite a entrada e saída dos dados do sistema.

- a "RETURN STACK", ou pilha de endereços de retorno, é a usada para o encadeamento executivo das funções, guardando os endereços de retorno. É mais comumente chamada de pilha de retorno, já familiar das sub-rotinas acionadas por "CALLA sub".

Em alguns sistemas exitem outras pilhas, destinadas a auxiliar processos numéricos ou para aumentar a flexibilidade dos laços DO-LOOP, ou mesmo para criar instruções especiais.

No Miniscript, além das duas pilhas já citadas, são utilizados outras duas, a saber:

- "INDEX STACK", ou pilha de índices. Consta de uma pequena pilha, especialmente destinada a gravar os índices correntes de laços DO-LOOP reentrantes. Maiores detalhes no capítulo sobre estruturas de controle.

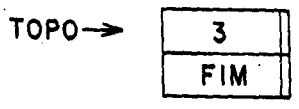
- "AUXILIAR STACK", é uma pequena pilha especialmente destinada a auxiliar a troca de dados em algumas funções numéricas e gráficas. Não é acessível pelo usuário diretamente.

O acesso às pilhas é feito de forma diferenciada.

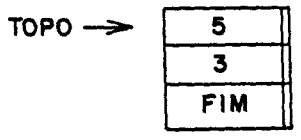
O "PARAMETER STACK" é a pilha cujo acesso é automatico, sempre que houver algum intercâmbio de informações entre funções. Normalmente, o dado que será imediatamente utilizado está sempre no "Topo da Pilha".

Por exemplo:

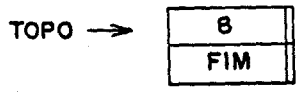
O usuário deseja somar dois números, "3" e "5".
Primeiramente ele introduz o número "3", tal número é processado e colocado no "topo da pilha".



A seguir o usuário introduz o outro número , "5".
Tal número é colocado no topo da pilha, e o número "3" é "empurrado" para baixo:



Em seguida o usuário coloca a operação desejada "+".
A função "+" foi definida no dicionário da seguinte forma: "pega o elemento no topo da pilha, soma com o segundo elemento da pilha e coloca o resultado no topo da mesma."
Após a execução, ter-se-á no topo da pilha o resultado.



Não há limite, a princípio, para o número de elementos colocados na pilha, porém não se deve ultrapassar certos limites, por segurança. Ou seja, o sistema prevê um espaço razoável para a maioria das aplicações e caso tal espaço seja ultrapassado o sistema indica a condição de erro (Stack Overflow). Outra situação de erro muito comum é tentar retirar da pilha, através de alguma função, elementos que ela não possui, ou seja, tirar mais elementos do que foram colocados. Neste caso, o sistema sinaliza erro "Stack Underflow".

Outro fato digno de atenção foi a ordem dos operandos numa função aritmética . Pelo exemplo acima, a operação desejada pelo usuário foi colocada após os elementos que fariam parte, ou seja:

3 5 +

Ao invés de:

3 + 5 =

Tal forma é chamada de Notação Poloneza Reversa.

A maioria das linguagens TIL (como as calculadoras HP) usam de tal notação pois ela simplifica muito a estrutura do processamento, sendo consequência direta do uso das pilhas.

Todas as funções nos TIL, que utilizam da pilha, buscam seus dados nesta, usando tal convenção de notações de operações.

Dessa forma, o usuário deve se certificar que os dados necessários foram colocados na pilha na ordem e no número que a função desejada os usa. Se, por exemplo, no exemplo anterior se quisesse a operação "-"(subtração) o resultado de:

3 5 -

seria "-2" colocado no topo da pilha. Se a seguir fosse introduzida uma nova operação , por ex: "+", o sistema iria buscar dois elementos na pilha que por sua vez somente contém um. O sistema, neste caso, sinalizaria a condição de erro.

Existe uma série de funções especialmente destinadas a manipular os dados na pilha, ou seja, alterar a ordem e as vezes o valor dos dados colocados nela.

Por exemplo, eis algumas: (os símbolos antes e depois simulam os seus efeitos na pilha).

x DUP	x x	Copia o topo da pilha
y x DROP	y	Elimina o elemento do topo da pilha
y x SWAP	x y	Intercambia os elementos no topo da pilha
y x OVER	y x y	Copia o segundo elemento no topo

No apêndice estão outras funções da mesma classe.

O "return stack" ou pilha de endereços de retorno , é acessado continuamente pelo sistema para manter o encadeamento executivo das funções. Normalmente o usuário não necessita chamá-lo, ou acessá-lo. Porém , caso se deseje , existe um conjunto de operadores que permitem a manipulação de dados através dessa pilha. O usuário deve , no entanto , manter um controle apurado nessa operação, pois o sistema pode perder o encadeamento se ocorrerem erros.

Como exemplo, existem a função "R>" (to-R), que transfere o topo da pilha de retorno para a pilha de parâmetros e a função "R<" (from-R), que faz o inverso.

O "index stack" será analizado quando se introduzirem as estruturas de controle DO-LOOP.

O "auxiallary stack" é de uso exclusivo do sistema, não tendo o usuário acesso.

As pilhas tem sempre o tamanho de 32 bits na implementação corrente, servindo indistintamente para armazenar números, strings, endereços, etc. A pilha não checa se o dado nele armazenado é consistente com o tipo esperado pela função que o usará.

As pilhas são utilizadas para a transferência ou armazenamento temporário de dados entre funções. Porém , as vezes é necessário que certos dados fiquem armazenados para uso posterior, independentes da movimentação da pilha de parâmetros. Estes elementos armazenadores são chamados variáveis.

V.6.)- Variáveis, Constantes

As variáveis são elementos que permitem o armazenamento de dados ou resultados, para serem usados posteriormente.

Através de um exemplo será analisado o seu funcionamento.

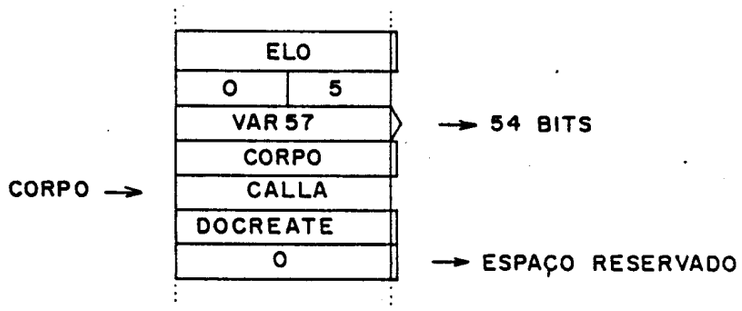
O usuário deseja criar uma variável, chamada por exemplo VAR57.

O usuário deve inserir:

O VARIABLE VAR57

Após tais comandos, no dicionário existe uma nova função chamada VAR57, possuindo a mesma estrutura de todos as entradas no dicionário. O comando VARIABLE cria a tal "função" VAR57 e insere no corpo desta uma chamada a uma rotina interna do sistema de nome DOCREATE . Reserva também no dicionário, logo a seguir, um espaço vazio que será usado como local de armazenamento. O zero é o valor inicial desejado.

No dicionário, uma variável tem a forma mostrada na figura abaixo:

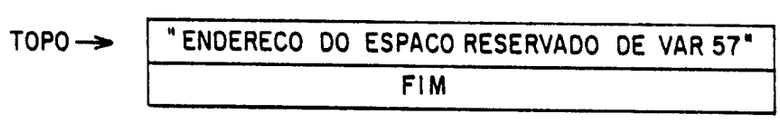


Quando o usuário chama a "função" VAR57 a execução cai no corpo desta e executa a linha "CALLA DOCREATE". A função "DOCREATE" nada mais faz do que colocar o endereço do espaço reservado no topo da pilha.

O espaço reservado é sempre a célula de 32 bits, imediatamente seguinte ao "CALLA DOCREATE". Assim o que DOCREATE faz é somente copiar o topo da pilha de retorno no topo da pilha de parâmetros.

Assim , quando o usuário insere:

VAR57



No topo da pilha está o endereço de tal variável.

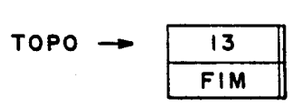
Para se colocar ou retirar dados deste endereço usa-se as funções "@" (fetch) e "!" (store).

A função "@" (fetch) pega o número (que é um endereço) que está no topo da pilha e copia o conteúdo deste endereço novamente no topo da pilha.

Supondo que VAR57 contém o número 13, para recuperá-lo, o usuário insere:

VAR57 @

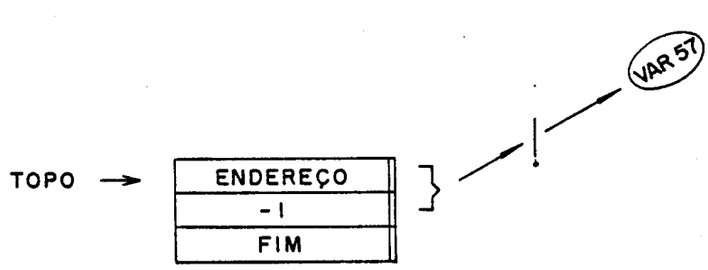
após o processamento o número "13" é colocado na pilha:



A função "!" (store) faz o contrário, pega o topo da pilha e o usa como sendo o endereço que será armazenado o segundo elemento da pilha.

Por exemplo, o usuário deseja armazenar "-1" em VAR57:

-1 VAR57 !



Em VAR57 estará armazenado -1.

Com a utilização correta das funções "@" (fetch) e "!" (store) é possível a implementação de endereçamento indireto, relativo, indexado ,etc . Por exemplo:

```
VAR57  @  @
```

ou

```
325 VAR57  @ !
```

Um tipo especial de variável são as constantes.
Por exemplo, o usuário deseja criar uma constante :

```
12  CONSTANT  DUZIA
```

Neste caso, será colocado no dicionário uma função chamada "DUZIA". Quando a função "DUZIA" é executada, coloca-se no topo da pilha o valor 12.

Internamente a função DUZIA é uma variável com a função "@" (fetch) embutida.

Tanto as variáveis quanto as constantes , como as pilhas, armazenam valores de 32 bits . Tais valores podem ser endereços, números, flags, strings, etc. A distinção é feita pela função que irá usá-los. Ou seja, o mesmo número pode para uma função ser um endereço, para outra um "flag" , etc.

Normalmente, por convenção, os números são considerados como codificados em "complemento de dois" e sempre inteiros.

Para a manipulação desses números existem uma série de funções aritméticas e lógicas que serão mostradas a seguir.

V.7)- Algumas Funções Aritmeticas e de Comparação

Similarmente às linguagens TIL geralmente encontradas , o sistema aqui implementado somente permite o uso de aritmética inteira. A implementação de operações em ponto flutuante é perfeitamente possível , como encontrada em algumas versões da linguagem FORTH (Lutus,83).

O Miniscript , para manter a simplicidade do código e a rapidez característica das implementações "inteiras" das "TILs", considera os números como sempre inteiros e em "complemento de dois". O sistema também não verifica condições de "OVERFLOW", quando um número ultrapassa a máxima capacidade de 32 bits , truncando-o. Cabe ao usuário verificar se sua aplicação corre o risco de entrar nesta situação e tomar as precauções devidas.

O fato de o Miniscript não possuir, pelo menos em seu núcleo básico, uma aritmética "real" não configura uma "desvantagem" , pois sendo sua finalidade básica a de controlar um dispositivo impressor, não é necessário que seja intrínsicamente capaz de calcular funções como "seno" , "logarítmo" etc. Mesmo em aplicações gráficas , a aritmética inteira é suficiente na maioria dos casos , visto que a estrutura "raster" por si só não permite "pontos fracionários". Mesmo nas definições de caracteres gráficos é possível a definição de contornos somente usando funções "inteiras" . Se o usuário necessitar, no entanto,

tais funções podem ser perfeitamente implementadas , seguindo os vários esquemas propostos em aritmética computacional. Existem, também co-processadores, tais como o TMS34082, que facilitam a implementação (Texas,88).

O núcleo do Miniscript contém uma série de funções seguindo o acima exposto. Algumas funções :

x y + z soma (x+y=z)

x y - z subtração (x-y=z)

x y * z multiplicação (x*y=z)

x y / q divisão inteira (q é quociente de x/y)

x y MOD r divisão inteira , apresentando o resto de x/y

x y /MOD r q divisão inteira , apresentando o quociente e resto

x y z */MOD r q
multiplica x*y e divide por z, usando 64 bits na operação intermediária interna . apresenta o quociente e resto.

x ABS y valor absoluto , se x<0 y=-x , se x>0 y=x

x NEGATE y y=-x

Como está somente disponível a aritmética inteira, torna-se interessante a utilização de razões inteiras na implementação de certas funções. Dessa forma algumas aproximações podem ser usadas para a representação de números reais, tais como :(erro < 0.0000001)

número PI (3.14159)	aproximação	355/113
número e (2.71828)	aprox.	25946/9545
Raiz de 2 (1,41421)	aprox.	27720/19601
Raiz de 3 (1,73205)	aprox.	32592/18817
Raiz de 10 (3,16227)	aprox.	27379/8658

No apêndice são apresentadas mais funções aritméticas.

Usando as funções aritméticas ,muitas vezes é necessária a comparação de dois números para que o programa tome alguma ação dependendo do resultado. Tais funções são chamadas de funções relacionais e as linguagens TIL possuem em grande número. O resultado dessa função são sempre "flags", ou sinalizadores que assumem o valor TRUE(-1) ou FALSE(0) no topo da pilha.

Algumas funções implementadas no Miniscript:

x y < flag Retorna flag=TRUE no topo da pilha, se x<y caso contrário, flag=FALSE.

```
x y > flag      retorna flag=true se x>y

x y = flag      retorna flag=TRUE se x=y

x 0< flag       retorna flag=TRUE se x<0

x 0> flag       retorna flag=TRUE se x>0

x 0= flag       retorna flag=TRUE se x=0
```

No apêndice existe a relação completa de tais funções.

As funções relacionais são utilizadas, principalmente, para desviarem o fluxo do programa, dependendo dos resultados de certa operação. As funções que usam dos resultados e perfazem tais desvios são chamadas de funções ou estruturas de controle.

V.8)- Algumas Funções ou Estruturas de Controle

As "funções de controle" são as estruturas que permitem um controle sobre fluxo de processamento. As linguagens "TIL" normalmente são fortemente estruturadas, consequência de sua organização interna, onde todas as funções e definições são modulares.

Por serem modulares, uma determinada aplicação ou função desejada pelo usuário é montada reunindo os módulos necessários e já definidos anteriormente.

As estruturas de controle devem também ser modulares. Existem várias formas de implementação dessas estruturas, sendo que a maioria geralmente age durante a montagem no dicionário da função, na qual faz parte (Loelinger,81)(Ting,86).

No Miniscript as estruturas de controle agem similarmente, atuando na forma como uma definição iniciada por ":" (colon) está sendo montada no dicionário. Assim, as estruturas agem durante o modo de compilação "montando" as funções para que no modo interpretativo e na execução tomem as decisões necessárias.

No Miniscript estão disponíveis, entre outros, as seguintes estruturas de controle.

```
flag IF {  executa se Flag = True } THEN

flag IF {  executa se Flag = True } ELSE
        { executa se Flag = False } THEN

BEGIN { cláusula a ser repetida se flag = False } flag UNTIL

BEGIN  {  cláusula  a ser repetida} flag WHILE
       {  cláusula  a  ser repetida se Flag = False } REPEAT

n 1 DO { cláusula a ser repetida ,
         de 1 até n incrementando 1 } LOOP
```

Por exemplo:

O usuário já definiu anteriormente duas funções, dentro do vocabulário contido em "CURRENT", chamadas "GIRADIREITA" e "GIRAESQUERDA" em uma determinada aplicação.

O usuário deseja agora criar a função "MEXE" que perfaz o seguinte:

a)- pega o número que está no topo da pilha;

b)- testa se o número é positivo;

c)- se positivo executa GIRADIREITA, caso contrário GIRAESQUERDA

Para tal função o usuário pode escrever:

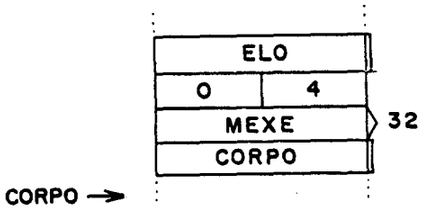
```
: MEXE  0> IF GIRADIREITA  ELSE  GIRAESQUERDA  THEN;
```

Para a utilização basta, por exemplo:

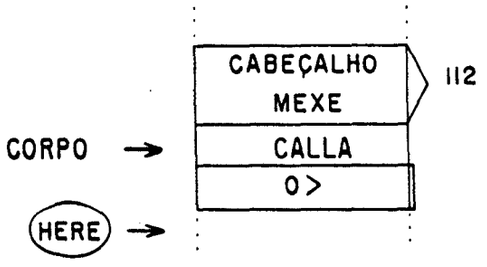
```
-20  MEXE          (aciona GIRAESQUERDA)

 31  MEXE          (aciona GIRADIREITA)
```

Acompanhando desde o início, como o sistema montou a função MEXE, várias considerações poderão ser feitas.

O sistema ao encontrar a função ":" (Colon) entra no modo compilativo e cria um cabeçalho (header) no dicionário, como já mostrado nos capítulos anteriores:

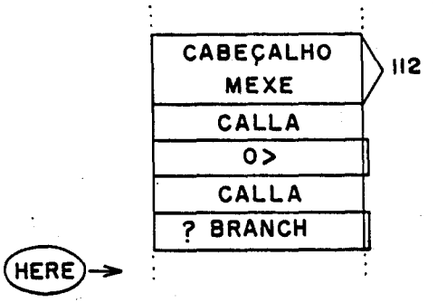


O sistema encontra a função "0>" e a compila no dicionário. Observe-se que o ponteiro do dicionário (HERE) vai se incrementando automáticamente, sempre apontando para a próxima posição livre no dicionário:

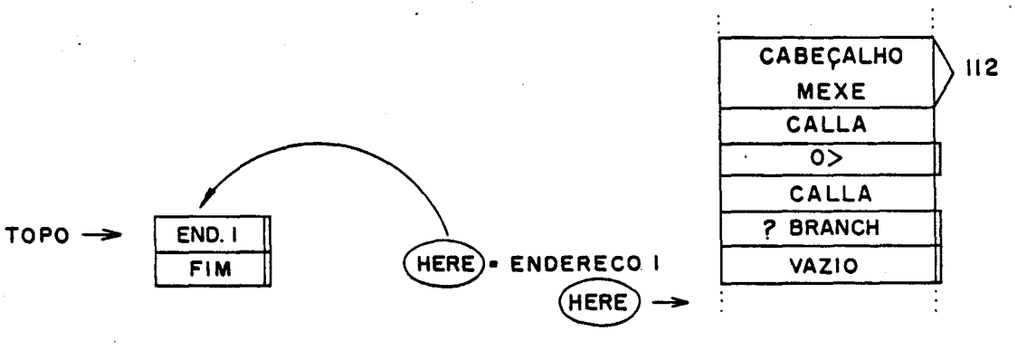


O sistema encontra, então, o comando IF. Como se esta no modo compilativo, a rotina FIND faz sua busca além de pelo vocabulário contido em CURRENT, também pelo vocabulário COMPILER. Neste vocabulário estão contidas as funções que são somente disponíveis no momento em que se cria uma nova função ou "entidade" e são as responsáveis pelo comportamento do sistema no modo compilativo. Neste vocabulário existe a função IF, que faz o seguinte:

Primeiramente copia no dicionário ,na próxima posição vaga (indicada por HERE); a instrução "CALLA ?BRANCH"



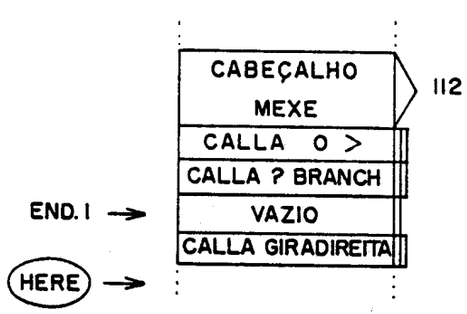
A seguir, copia o valor atual de HERE (que será chamado aqui de Endereço-1) no topo da pilha e , em seguida, incrementa HERE de forma que apos ?BRANCH exista um espaço vazio, suficiente para armazenar um endereço (32 bits)



E importante salientar que a função "IF" é "executada" durante o modo compilativo, ao invés de ser "compilada" no dicionário (como a função imediatamente anterior "0>"). Tal comportamento advém do fato de que a maioria das funções do vocabulário "COMPILER" possuem o atributo de serem "IMEDIATAS". Funções imediatas são aquelas que são executadas mesmo no modo compilativo, pois possuem o "precedence bit" , ou bit de precedência , dentro do cabeçalho da função, ajustado de acordo.

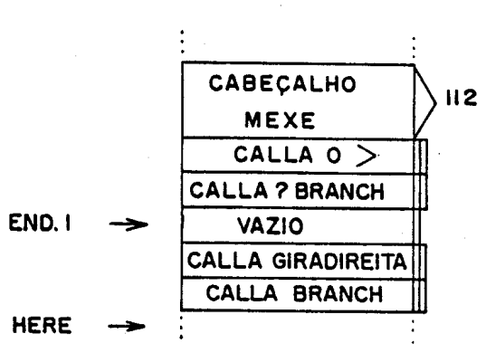
Observa-se, também , que no topo da pilha está armazenado temporariamente o endereço da célula vazia.

A seguir o sistema encontra o comando GIRADIREITA, que após a busca pelo vocabulário contido em CONTEXT compila no dicionário de forma similar ao já apresentado.

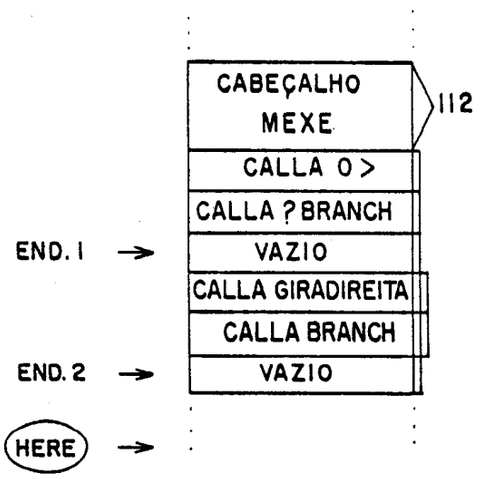


O sistema, continuando a compilação encontra o comando ELSE, que também pertence ao vocabulário COMPILER e possui o atributo de "imediata", não sendo compilado .

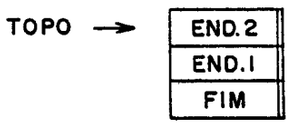
A função ELSE faz o seguinte: copia na próxima posição vaga a instrução "CALLA BRANCH":(obs: BRANCH não é igual a ?BRANCH)



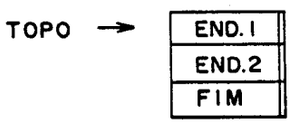
Em seguida copia o valor atual de HERE no topo da pilha (que será chamado de Endereço-2) e a seguir reserva outro espaço vazio (32bits).



A pilha apresenta a seguinte situação:

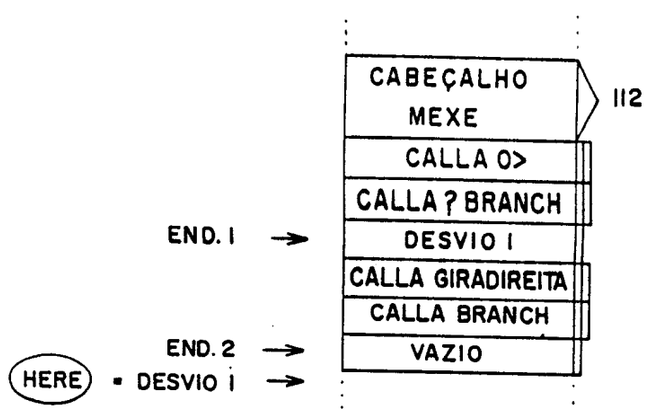


Continuando a execução da função ELSE, esta pega e troca os dois elementos no topo da pilha (SWAP).

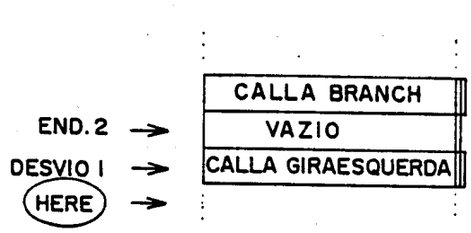


A seguir copia o valor atual de HERE (que será chamado "desvio1") no endereço contido no topo da pilha, isto é, endereço-1.

Como já salientado, o endereço-1 é o endereço da célula vazia no dicionário, logo apos CALLA ?BRANCH.

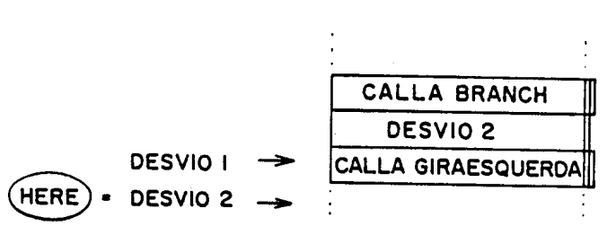


O sistema continua agora a compilação, no caso GIRAESQUERDA.



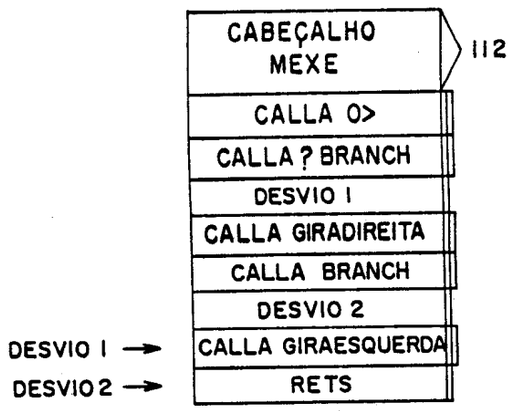
O sistema encontra finalmente as instruções THEN, pertencendo ao vocabulário COMPILER.

Esta função simplesmente copia o valor atual de HERE (que será chamado de Desvio-2) no endereço contido no topo da pilha, no caso, Endereço-2.



A seguir o sistema encontra a função ";" (semi-colon), também do vocabulário COMPILER , que insere a instrução "RETS" no dicionário em HERE e põe o sistema novamente no modo interpretativo.

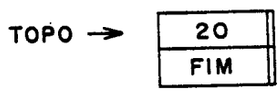
Observando o dicionário , pode-se agora explicar o funcionamento da função MEXE.



No exemplo o usuário introduzia:

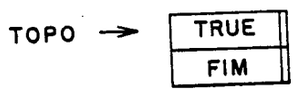
20 MEXE

O sistema, agora no modo interpretativo, pega o número "20" e coloca na pilha.

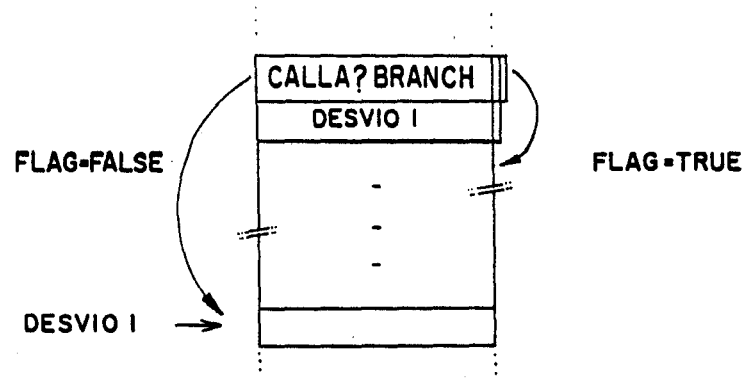


A seguir executa a função MEXE. Dentro desta , a primeira instrução é "CALLA 0>" que, como visto, pega o número que está no topo da pilha e testa se tal número é menor que zero. Se afirmativo, coloca no topo da pilha o valor TRUE (-1), caso contrário FALSE (0), como sinalizador (flag).

No exemplo:



A próxima instrução é CALLA ?BRANCH. Tal instrução, que foi colocada por IF, faz o seguinte: pega o sinalizador que está no topo da pilha e verifica se é igual a FALSE. Se for, o processamento desvia para o endereço contido na próxima célula do dicionário (que contém Desvio-1). Caso contrário, pula a próxima célula e continua o processamento.



No exemplo ,como 20 > 0 , flag = TRUE, então o processamento pula sobre a célula que contém "Desvio-1" e executa CALLA GIRADIREITA.

A seguir, encontra a instrução "CALLA BRANCH" que nada testa,(desvio incondicional), desviando automáticamente o processamento para o endereço contido na próxima célula, "Desvio-2".

No endereço "Desvio-2" se encontra a função "RETS" que termina a execução da função MEXE.

Como no exemplo acima , pode-se constatar que as estruturas de controle "IF-THEN-ELSE", "BEGIN-UNTIL", "BEGIN-WHITE-REPEAT" seguem o mesmo padrão de montagem e execução , todos eles se utilizando das funções internas ?BRANCH e BRANCH. É interessante notar que, apesar da linguagem ser estruturada na forma como o usuário a introduz , internamente ela se utiliza de comandos tipo "go-to", visando economia de tempo de execução.

Como se pode observar , através de conceitos simples são possíveis de se criar estruturas complexas de processamento. Por se utilizarem das pilhas, permitem o encadeamento de estruturas, tais como no exemplo abaixo:

```
┌IF
│   ┌IF
│   │   ┌IF
│   │   └THEN
│   │ ELSE
│   │   ┌IF
│   │   └THEN
│   └THEN
└THEN
```

Obviamente tais estruturas devem ser usadas em pares e devem ser consideradas como delimitadoras de estruturas de decisão dentro das funções sendo compiladas por ":" (Colon). Ou seja, para cada IF deve haver um THEN, para cada BEGIN um UNTIL ou REPEAT.

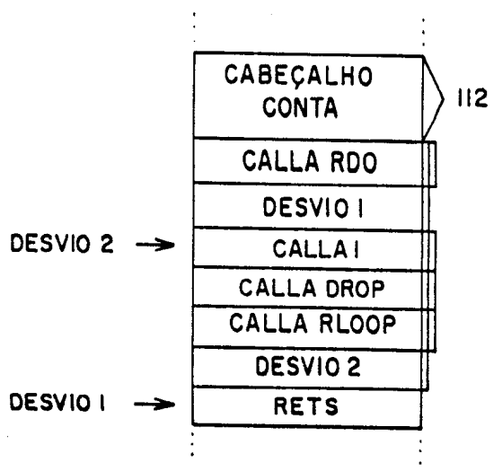
O sistema não verifica o cruzamento de estruturas que possam causar erros (por exemplo: IF BEGIN THEN). Normalmente o usuário deve se certificar disso. Porém, existem TILs que sinalizam a ocorrência de tais cruzamentos durante o modo de compilação. (Ting,86).

Os laços de repetição DO-LOOP são construídos de forma similar , porém ao invés de se utilizarem das funções ?BRANCH e BRANCH, se utilizam das funções internas RDO e RLOOP.

Um exemplo:
O usuário criou para alguma aplicação fictícia a função CONTA, definida da seguinte forma:

```
: CONTA  DO
             !  DROP
         LOOP
;
```

Após a compilação no dicionário tem-se:



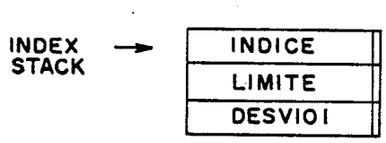
Na execução o usuário deve fornecer dois números, como no exemplo:

5 2 CONTA

Na execução a função interna RDO pega o topo da pilha e o considera como sendo o índice inicial do LOOP, (2).

O segundo valor da pilha é o limite final, onde o LOOP deve ser interrompido.(5). Em seguida, pega o endereço contido na próxima célula (Desvio-1) e junto com o índice e o limite, são colocados no "!NDEX STACK" (pilha de índices).

Esta pilha, como já definido, é de pequenas dimensões e existe exclusivamente para a guardar os elementos necessários ao funcionamento dos laços DO-LOOP.



A seguir o sistema executa a função "I", pertencente ao vocabulário CORE, que tem a finalidade principal de copiar o topo da pilha de índice no topo da pilha de parâmetros.

Tal função visa a copiar o índice corrente num laço DO-LOOP, para o uso do usuário.

A seguir, encontra-se nadefinição a função "DROP", já conhecida, que elimina o topo da pilha de parâmetros.

Em seguida tem-se a função "RLOOP". Tal função foi posicionada no dicionário pela compilação de LOOP. Na mesma execução RLOOP pega o topo da pilha de índices (no caso o índice), incrementa de 1 e verifica se é maior que o segundo elemento desta pilha (o limite). Se for menor a execução desvia para o endereço contido na próxima célula do dicionário, no caso "Desvio-2".

Caso o índice ultrapasse o limite, RLOOP elimina os três elementos no topo da pilha de índices e a execução pula a próxima célula , no caso executando a instrução RETS.

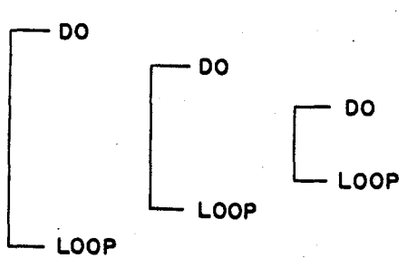
A razão pela qual na pilha de índices é armazenado o endereço "Desvio-1" é a de possibilitar o uso do comando "LEAVE", que cancela um laço DO-LOOP. Na execução de LEAVE, este simplesmente faz o processamento pular para o endereço contido na terceira posição da pilha de índices.

Por exemplo:

```
: NUNCACONTA DO I DROP LEAVE LOOP;
```

Como as funções DO LOOP se utilizam de uma pilha auxiliar é possível o encadeamento de laços DO-LOOP.

Por exemplo:



No caso de laços DO-LOOP reentrantes, a cópia dos índices mais externos também é possível. Como visto , a função "I" tem a finalidade de copiar o topo da pilha de índice no topo da pilha de parâmetros. Dessa forma, tal função copia o índice do Laço DO LOOP mais interior. Existem outras funções para manipular a pilha de índices, para permitir a cópia dos índices de laços mais exteriores.

No apêndice existem comandos que permitem construir outros tipos de laços repetitivos.

Além das estruturas de controle, uma linguagem , para ser completa deve possuir mecanismos para se comunicar com o ambiente externo e o usuário . Tais comandos são as funções de entrada/saída.

V.9)- Algumas Funções de Entrada/Saída

As linguagens TILs, por serem fortemente interativas, possuem uma série de comandos especificamente destinados a agilizar a transferência de informação entre a linguagem e o usuário. Algumas TILs, como o FORTH, possuem até editores embutidos na linguagem.

No caso de implementação aqui discutido, como o seu objetivo é o de controlar dispositivos impressores, o conjunto de comandos que perfazem as funções de entrada/saída podem ser divididos em duas partes principais.

O primeiro conjunto, é o que faz a comunicação entre a linguagem e a impressora, no que tange a geração de imagem a ser impressa. Tal parte será discutida no capítulo das funções gráficas.

O segundo conjunto, é o que faz a comunicação entre a linguagem e o usuário ,ou sistema ,que lhe envia as informações que devem ser impressas.

A comunicação entre os dispositivos impressores e o seu sistema hospedeiro normalmente é feito por canais padronizados , tais como as interfaces RS232, 422, etc.

Tais interfaces, no entanto, não são colocadas de forma padronizada dentro do hardware impressor. Dessa forma, o sistema deve possuir rotinas que saibam acessar e administrar tais posições no "hardware", para efetivar as comunicações desejadas.

Todas as estruturas e instruções até aqui descritas do Miniscript são independentes do "hardware impressor" utilizado, o que caracteriza a linguagem como sendo "Device Independent". Porém, como o sistema deve saber se comunicar com o "Hardware" são previstas algumas instruções "pontes".

No caso são duas: "INPUT" e "OUTPUT".

A função INPUT tem a finalidade de receber os dados e comandos pelo canal de comunicação e armazená-los numa memória acumuladora ou "buffer" chamado de "TIB", ("Text Input Buffer" ou acumulador de dados de entrada). A rotina INPUT quando solicitada vai até o canal de comunicação e pega os dados e comandos que ali chegaram, fazendo o "handshake", ou sinalizações necessárias para que a comunicação se efetue. A rotina INPUT vai acumulando os dados recebidos pela interface em "TIB", até que encontre o fim deste ou o comando "Carriage Return".

A função OUTPUT tem a finalidade oposta de INPUT. Essa função pega os dados acumulados em um "Buffer" chamado de "TOB", ("Text Output Buffer") e os encaminha à interface de comunicação, (quando o hardware permitir esta posibilidade).

Vale a pena ressaltar que as funções INPUT, OUTPUT, ABORT e SHOWPAGE (estas duas últimas serão mostradas futuramente) são as únicas funções "dependentes do hardware" de todo o sistema. Ou seja, são as únicas cujo código implementado depende da disposição física-eletrônica da máquina que será utilizada. Tal fato confere ao design da implementação aqui proposta grande grau de portabilidade , pois somente em 4 instruções devem ocorrer modificações quando se troca de "hardware".

Ressalta-se que o uso dessas instruções pelo sistema é sempre o mesmo, ou seja, o usuário não necessita mudar o seu aplicativo.

O sistema manipula os dados em TIB e TOB, através de um conjunto de comandos.

Por exemplo: um número é colocado em TIB pela função INPUT, tal número foi transmitido em caracteres ASCII, como de praxe. Para transformar este conjunto de caracteres no número binário que ele representa existe a função NUMBER. Esta função transforma os caracteres ASCII segundo a base de numeração escolhida, armazenando na variável BASE.

Se BASE=10, determina base decimal.

Se BASE=16, determina base hexadecimal, onde as letras A, B, C, D, E e F tem significado. O usuário pode adotar a base que desejar e ela é válida para todas as operações de entrada/saída subsequentes.

Por exemplo:

```
8 BASE ! (O usuário escolheu a base octal)
```

Usando BASE, NUMBER gera o número binário correspondente, que será armazenado no topo da pilha.

A função "." (dot) faz o inverso, pega o número (binário) que está no topo da pilha, converte-o para string equivalente usando BASE, e o coloca em "TOB".

A função TYPE pega um string na memória, cujo endereço está no topo da pilha e o transfere para "TOB", acionando OUTPUT.

No apêndice consta uma série de funções auxiliares para as operações de entrada/saída.

V.10)- Criando Novas Estruturas de Dados

Na implementação do Miniscript, são chamadas de "estruturas" as funções, ou entidades, que quando utilizadas criam certos tipos de estruturas de dados .

Já foram discutidas, até agora, algumas estruturas comuns às linguagens TIL e colocadas nessa implementação, tais como: VOCABULARY, VARIABLE, CONSTANT.

Porém, o usuário pode sentir a necessidade, para determinada aplicação, de uma nova estrutura.

Por exemplo:

O usuário deseja criar a estrutura "ARRAY", para definir matrizes unidimensionais.

Após definida a estrutura "ARRAY" o usuário poderá criar matrizes, como no exemplo abaixo:

```
30 ARRAY MATRIZA
```

O usuário criou a matriz MATRIZA de dimensão 30 (numerados de 0 a 29) . Se o usuário quer armazenar o número -27, na posição 5 de MATRIZA, ele deverá comandar como abaixo:

```
-27 5 MATRIZA !
```

Se o usuário quer recuperar o conteúdo da posição 5 da MATRIZA deverá:

```
5 MATRIZA @
```

Como se pode observar, o comando "ARRAY" criou um novo elemento, por isso "ARRAY" é uma nova estrutura disponível na linguagem.

No exemplo, o usuário pode definir o comando ARRAY da seguinte forma:

```
: ARRAY
    <BUILDS SWAP ALLOT
    DOES> 32 * R> +
;
```

A função ARRAY é compilada normalmente no vocabulário contido em CURRENT, porém possui algumas particularidades.

Os comandos <BUILDS e DOES> servem para separar dois trechos dentro da definição de ARRAY, trechos esse que tem comportamento distinto.

Os comandos entre <BUILDS e DOES> são compilados e são "executados" quando se executa a função ARRAY, no exemplo , produzindo a matriz MATRIZA.

Os comandos entre DOES> e ";" (semi-colon) são compilados e não são executados quando se executa ARRAY. Somente serão executados quando o elemento produzido pela função ARRAY for executada, ou seja, quando executar-se MATRIZA.

A função ARRAY é chamada função "pai" de MATRIZA.

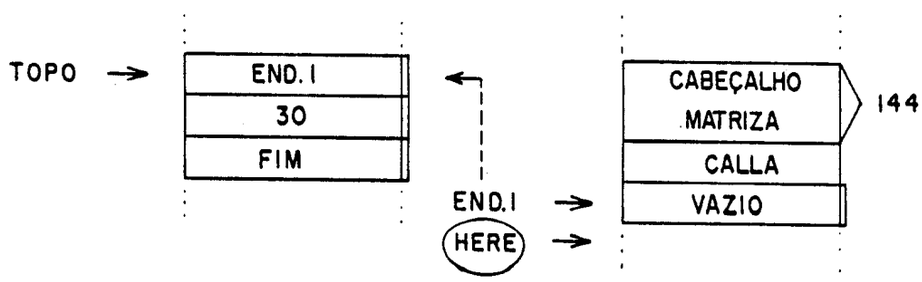
Agora, pode-se acompanhar o ocorrido quando o usuário entrou com:

```
30 ARRAY MATRIZA
```

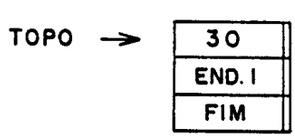
Primeiramente o número 30 foi para o topo da pilha. A seguir, o sistema executa a "função" ARRAY.

Em ARRAY encontra-se, primeiramente, o comando <BUILDS.

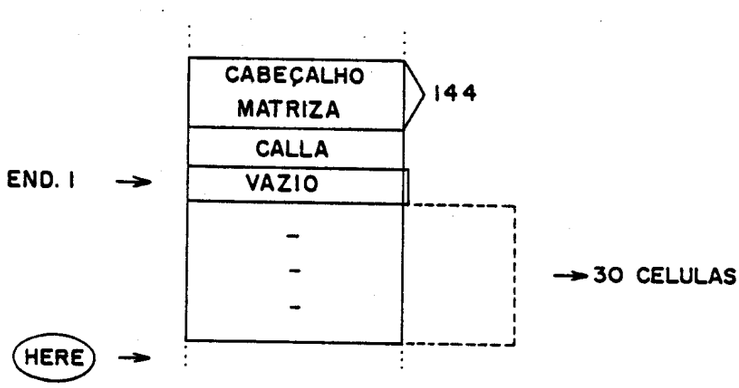
Este comando cria um cabeçalho com o nome que vem a seguir (MATRIZA). Logo após, insere no corpo da nova definição uma instrução "CALLA" e reserva uma célula vazia (32bits), registrando seu endereço na pilha e incrementando HERE.



A seguir, através do comando SWAP, intercambia-se o topo da pilha:



O comando ALLOT pega então o número que está no topo da pilha e reserva este número de células (de 32 bits) no dicionário.

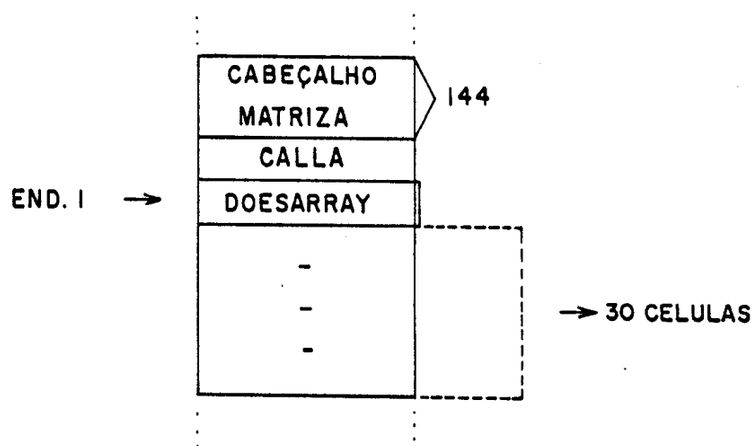


A seguir o sistema encontra o comando DOES>. O endereço fisico que a função DOES> se encontra dentro de ARRAY é importante pois indica a fronteira entre as duas regiões de processamento distintas em ARRAY. O endereço da próxima função após DOES> será chamado de DOESARRAY.

A tarefa que DOES> executa é a de simplesmente colocar o endereço DOESARRAY no local apontado pelo endereço que está no topo da pilha. Ou seja, copia DOESARRAY em endereço-1.

A seguir, a execução de "ARRAY" é terminada. Nota-se que os comandos "32 * R> +" não foram executados.

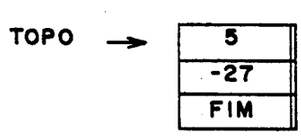
O dicionário finalmente apresenta o seguinte aspecto:



Pode-se verificar agora o que ocorre quando se executa:

-27 5 MATRIZA !

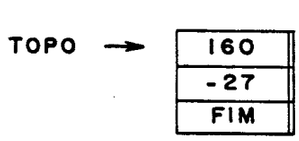
O número -27 vai para o topo da pilha e logo a seguir o número 5:



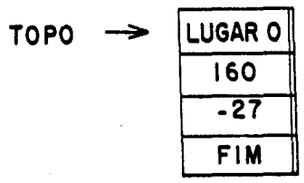
O sistema encontra a "função" MATRIZA.

A função "MATRIZA" é executada e em seu corpo é encontrado um CALLA DOESARRAY. Assim, a execução pula para o endereço "DOESARRAY" , trecho que está dentro da definição de ARRAY e contém: " 32 * R> + "

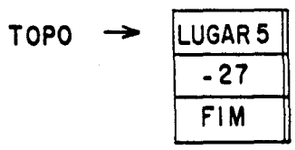
Neste local o comando 32 * simplesmente multiplica o topo da pilha por 32:



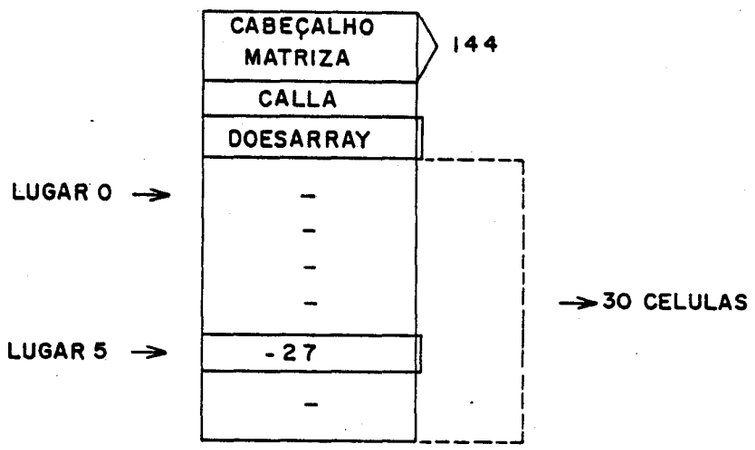
O comando R> transfere o topo da pilha de retorno para o topo da pilha de parâmetros. O topo da pilha de retorno, como se sabe , guarda o endereço de retorno da última instrução CALLA executada. Neste caso, é o endereço logo a seguir de CALLA DOESARRAY, ou seja, o endereço da primeira célula vazia. Tal endereço será chamado de "Lugar 0".



Por fim, encontra-se a operação "+" que soma "Lugar 0" com 160, cujo resultado é o endereço da célula 5 de ARRAY, que será chamado de Lugar-5:



A execução encontra o "RETS" do ARRAY e retorna ao sistema, que encontra a seguir de MATRIZA o comando "!" (store). Como se pode imaginar , o número -27 será colocado no endereço "Lugar 5".



Com os comandos <BUILD e DOES> é possível criar uma série de novas estruturas de dados e até mesmo de controle. Tal fato aumenta muito a flexibilidade do sistema em se adaptar à necessidade do usuário.

Note-se que, no fundo, os comandos <BUILDS DOES> trabalham na forma que uma nova definição é montada no dicionário.

Existem uma série de funções auxiliares que efetuam tarefas similares durante a montagem de funções no dicionário.

A função "'" (Tick) copia na pilha o endereço da próxima função na linha de comando, sendo enviado ao sistema. Por exemplo:

```
' DUP
(copia na pilha o endereço de execução de DUP )
```

A função COMPILE, quando executada ,compila a próxima função no dicionário. Por exemplo:

```
COMPILE   SWAP
```

(insere no dicionário um "CALLA SWAP").

No apêndice existem as descrições de outras funções de mesma classe.

V-11) O Sistema de Controle ou Sistema Operacional

Até agora foram discutidas as estruturas principais , algumas funções e como tais elementos se combinam para permitir a implementação dos aplicativos desejados pelo usuário.

Porém , para que todas essas partes funcionem bem e para que o conjunto passe a ter a "personalidade" descrita até aqui , é necessário que exista um conjunto de regras de "comportamento" implementadas no sistema.

Essas regras , na verdade , constituem o "sistema operacional" que a linguagem tem em seu interior.

Nas linguagens TIL , o "sistema operacional" nada mais é que uma função definida com base nos comandos da propria TIL.

Esta "função" é acionada automáticamente quando o sistema é inicializado.

O fato de ser o "sistema operacional" escrito na própria linguagem , permite ao usuário modificá-lo ou "personalizá-lo" .

No Miniscript a rotina que perfaz a tarefa de ser o "sistema operacional" adotado é chamada de "INNER".

A "função operacional" INNER foi criada usando uma série de funções e recursos pré-existentes, e tem a seguinte definição.

```
: INNER
  BEGIN
     INPUT
     SPACE TOKEN
     BEGIN
          ENDBUF @
     WHILE
          STACK
          DEFINED
          MODE @
          IF COMPILATOR ELSE INTERPRETER THEN
          SPACE TOKEN
     REPEAT
  AGAIN
;
```

A primeira observação relevante é que INNER foi definido usando o comando ":" (colon) . Ou seja, INNER tem a mesma estrutura que qualquer outra.

Na definição encontra-se um BEGIN que irá formar com AGAIN , no fim da função, um ciclo iterativo sem fim. O comando AGAIN equivale a "FALSE UNTIL", tornando o ciclo "BEGIN --- FALSE UNTIL" sem saída. Daí, pode-se concluir que INNER é a "alma" do sistema , só "morrendo" se entrar em pane por uso abusivo de alguma pilha , estrutura de controle ou estouro de memória.

A seguir, encontra-se o comando INPUT , já conhecido , que ativa a interface de comunicação e vai acumulando os caracteres e sinais provenientes desta interface, até encontrar um "carriage return".

A função INPUT atualiza um sinalizador chamado ENDBUF que indica se o acumulador TIB está vazio.

Quando INPUT recebe um "carriage return" interrompe o armazenamento em TIB e libera a continuidade de processamento dentro de INNER.

As funções SPACE TOKEN servem juntas para explorar o acumulador TIB, separando os comandos. A função SPACE simplesmente coloca no topo da pilha o caracter ASCII "space" . Este caracter na pilha é armazenado e usado como delimitador por TOKEN.

TOKEN percorre TIB separando os comandos delimitados por este caracter (esta é a razão pela qual no Miniscript os espaços em branco são importantes). TOKEN pega o próximo "string" e o põe a disposição do sistema para analise futura, verificando também se o fim do acumulador foi atingido , atualizando ENDBUF.

Continuando o processamento , encontra-se novamente uma estrutura de controle , agora do tipo BEGIN-WHILE-REPEAT. Esta estrutura usa como parametro de controle o sinalizador ENDBUF. Em suma tal estrutura fica verificando se o fim do acumulador TIB foi atingido .

Dentro do laço , o comando STACK serve para verificar se alguma das pilhas está ultrapassando o seu limite pré-estabelecido, avisando o usuário neste caso.

A função DEFINED é a que faz a busca pelos vocabulários, sendo uma expansão do comando FIND já comentado. Primeiramente DEFINED busca o comando separado por TOKEN no vocabulário CONTEXT , a seguir no CURRENT , seguindo até o CORE. Se estiver no modo compilativo , busca também no vocabulário COMPILER. A função DEFINED deixa na pilha alguns sinalizadores que serão usados adiante, pelo compilador/interpretador.

Na linha seguinte , encontra-se uma estrutura IF-ELSE-THEN, controlada pela variável MODE , que guarda um sinalizador indicando o modo do sistema. Se estiver no modo compilativo , aciona-se COMPILATOR , caso contrário INTERPRETER.

O ciclo continua até que TIB se esgote , onde novamente é acionado INPUT.

Resumindo o exposto, a função INNER aciona o canal de comunicação , acumulando a entrada de dados . A seguir, separa os comandos enviados e procura-os pelo dicionário. Se estiver no modo compilativo , aciona o programa compilador , caso contrário o programa interpretador. Tal processo continua até que o acumulador fique vazio , onde nova entrada é solicitada.

V-12)- O interpretador e o compilador

Para a compreensão do funcionamento da implementação dos programas COMPILATOR e INTERPRETER é necessário verificar os sinalizadores deixados na pilha por DEFINED, dentro do INNER.

DEFINED pega da pilha o endereço no acumulador de comando, ou palavra, separado por TOKEN e faz a busca pelo dicionário , na forma já descrita;

- se encontrar em algum vocabulário , coloca na pilha o endereço de execução desta palavra e um sinalizador TRUE

- se encontrar e tal palavra possuir o atributo de "imediata" , coloca na pilha o endereço e o sinalizador "-1"

- se não encontrar em qualquer vocabulário , retorna o endereço do acumulador e o sinalizador FALSE.

O Interpretador usa estes dados da seguinte forma:

```
: INTERPRETER
    <>0  IF
                EXECUTE
          ELSE
                NUMBER
                MISSING?
          THEN
;
```

Como se pode verificar , o interpretador testa se o número no topo da pilha é diferente de zero, (FALSE) , significando que o endereço a seguir é válido.

Neste caso, aciona a função EXECUTE que pega o endereço no topo da pilha e "corre" a função contida neste.

Caso o sinalizador seja "FALSE" , o interpretador tenta verificar se a palavra ou "string" é um número, através da função NUMBER.

Esta função transforma o "string", usando a base numérica corrente armazenada em BASE . Se a transformação for possível e bem sucedida , é colocado na pilha o número binário correspondente, seguido de um sinalizador de "sucesso". Caso contrário , é devolvido o endereço do "string" seguido de "fracasso" .

MISSING pega este sinalizador e se "fracasso", imprime tal "string suspeito" seguido de mensagem de erro. Caso contrário, segue para o fim do ciclo.

O compilador funciona de forma similar:

```
: COMPILATOR
    DUP <>0 IF
              -1 = IF
                        CALLA C, ,
                   ELSE
                        EXECUTE
                   THEN
            ELSE
                   DROP
                   NUMBER
                   MISSING?
                   COMPILE  LIT
                   ,
            THEN
;
```

Pode-se observar que a estrutura básica de COMPILATOR é a mesma de INTERPRETER. A diferenca é que quando uma palavra é encontrada , somente é executada se possuir o atributo "imediato". Caso contrário, os comandos "CALLA C, , " colocam no dicionário a instrução "CALLA" do TMS34010 seguido do endereço de execução da palavra, inserida neste pelo comando "," (comma) .

Se a palavra não é encontrada no dicionário , como no interpretador , NUMBER tenta convertê-la . Se a transformação for bem sucedida, ao invés do número correspondente ir para a pilha , os comandos " COMPILE LIT , " colocam no dicionário a função LIT seguida de tal número. (A função LIT quando executada vai pegar um número no dicionário e colocá-lo na pilha).

As implementação dos comandos COMPILATOR e INTERPRETER são as adotadas pelo autor . Note que é possível que o usuário crie a sua própria versão , adequando-a a sua necessidade específica.

Por exemplo: o usuário pode criar um sistema personalizado , chamado MYSYSTEM , combinando as funções do sistema na forma desejada.

Para criá-lo, basta proceder como numa definição ":" (colon) normal:

```
: MYSYSTEM   -  -  -  -  ;
```

Para colocá-lo em ação , basta:

```
MYSYSTEM
```

Nota-se que a "função" MYSYSTEM estará rodando sob INNER interrompido . Se houver alguma pane e queda deste sistema , "ABORT" automáticamente reativará INNER. O sistema "INNER" é o "inner-most" , ou o mais interno da implementação.

V-13) Alguns comentários finais sobre o núcleo básico

Como o objetivo visado da linguagem Miniscript é a de ser um controlador de dispositivos impressores , muitos dos recursos normalmente disponíveis nas TIL não foram aqui implementados.

Nas TIL existem embutidos no sistema programas, tais como: editores , assembladores , metacompiladores etc , que no ponto de vista do autor não são prioritários no objetivo de ser um gerador de imagens a serem impressas.

Ressalta-se , no entanto , que tais recursos são facilmente implantáveis pelo usuário pois as estruturas básicas necessárias estão todas contidas no núcleo aqui exposto.

Maiores informações podem ser obtidas nas referencias (Ting,86)(Loelinger,81)(Brodie,83)(Derick,85).

VI) IMPLEMENTAÇÃO DAS ROTINAS BASICAS - NUCLEO GRAFICO

VI-1)- Introdução

Neste capítulo, serão descritos os operadores gráficos principais e as estruturas de dados correspondentes, que implementam o modelo gráfico adotado, já descrito.

Serão descritos os operadores principais, acompanhados de uma breve descrição de seu funcionamento e sua relação com o TMS 34010.

Exemplos práticos serão apresentados visando inter-relacionar os comandos e funções ao modelo gráfico.

Assume-se que o leitor já se familiarizou com o modelo gráfico adotado, bem como com a estrutura da linguagem em si, sendo de vital importância a perfeita compreensão da estrutura do dicionário, o significado dos comandos: : (Colon) ; (Semi Colon) . ' (tick) @ (fetch) ! (Store) e as estruturas de controle: DO ! LOOP IF ELSE THEN.

Toda a linguagem foi criada assumindo-se que os pixels são de 1 bit (branco ou preto), visto que primariamente a linguagem destina-se a impressoras ,porém os conceitos aqui apresentados permitem a expansão para pixels de mais de 1 bit, permitindo a utilização de cores.

VI-2) Grupos funcionais de operadores gráficos

Escolhido o modelo gráfico, os comandos foram criados de forma a implementá-lo, permitindo sua manipulação. Tais comandos seguem uma organização que pode ser analizada pela classificação da ação que o comando efetua. Esta classificação visa permitir um maior discernimento da estrutura da linguagem.

A classificação , a grosso modo, divide os comando gráficos em 4 grandes grupos:

- Definidores de Contexto
- Definidores de Forma
- Definidores de Ação gráfica
- Comandos auxiliares e Funções de Convoniência

a)Definidores de Contexto

Os Definidores de Contexto são os comandos destinados a definir as "entidades" que estarão a disposição dos comandos de ação gráfica e definidores de forma. Definem, também, alguns parâmetros adotados pelo usuário que serão usados futuramente nas operações gráficas subsequentes. Normalmente, os comandos de definição de contexto nada mais fazem do que armazenar os dados escolhidos pelo usuário nas variáveis internas de controle que serão usados pelos comandos de ação.

Como exemplo deste tipo de comandos definidores de contexto, pode-se citar: (todos serão detalhados mais adiante)

SCALEX	(Fator de escala eixo X)
SCALEY	(Fator de escala eixo Y)
ROTATION	(Rotação dos eixos)
SHEARX	(Deformação eixo X)
SHEARY	(Deformação eixo Y)
TRANSPARENCY	(Atributo de transparência)
ORIGIN	(Origem do sistema virtual de coordenadas)
CURSTENCIL	(Stencil corrente)
CURBRUSH	(Pincel corrente)
BOUNDARY, etc	(Bordas escolhidas)

Outros tipos de definidores de contexto são os comandos destinados a especificar as operações que serão efetudadas durante a execução das ações gráficas , tais como:

REPLACE	(Função REPLACE)
GAND	(Função AND Gráfica)
GNOT	(Função NOT Gráfica)
GPLUS, etc	

Alguns definidores de contexto, muito importantes, são aqueles relacionados com a forma que as trajetórias devem ser interpretadas:

DRAWMODE	(Seleciona modo de uso do pincel)
FILLMODE, etc	(Seleciona modo de preenchimento)

Neste grupo estão também algumas funções que especificam os parâmetros para as funções de auxílio ao usuário, ou funções de conveniência. São definidores de contexto especialmente criados para facilitar o uso dos comandos impressores de caracteres, tais como:

SELECTTFONT	(Seleciona nome do fonte de trajetórias)
SELECTBMFONT	(Seleciona nome do fonte do Bit-Map)

b) Definidores de Forma

Neste grupo estão os comandos destinados a simular e especificar os "pinceis" , "reguas" , "gabaritos" , "logotipos", que permitirão o desenho das formas e figuras que serão colocados nos stencils. Neste grupo estão os comandos para a traçagem de linhas , curvas, linhas interpolas por Bezier, etc.

A combinação desses comandos pormitem a criação das trajetórias, que podem definir os logotipos , desenhos personalizados e até mesmo os caracteres. Uma trajetória , por si só, não é um desenho , mas sim uma entidade definidora analítica da forma deste desenho .

Como exemplo de definidores de forma estão os comandos

LINETO	(Traça linha)
MOVETO	(Posiciona)
ARCTO	(Arco)
CURVETO , etc	(Bezier)

Os comandos definidores de forma, por si só, já efetuam ações gráficas, usando os parâmetros previamente selecionados nos comandos de definição de contexto. Assim, o efeito que um comando definidor de forma produz , pode ser alterado, bastando somente se modificar os definidores de contexto prévios. Um exemplo muito comum desta situação são os caracteres definidos como trajetórias, que podem ser expandidas, rotacionadas etc .

c) Definidores de Ação Gráfica

Neste grupo estão os operadores que efetuarão as ações gráficas entre os "stencils" . Os "stencils" possuem os atributos e parâmetros previamente selecionados pelos comandos definidores de contexto e também já possuem as formas devidamente definidas e colocadas.

Como exemplo, temos os comandos:

COMPOSE	(Ação gráfica)
PASTE	(Ação gráfica)
DETACH	(Ação gráfica)
REPASTE	(Ação gráfica)
TRANSFER	(Ação gráfica)

d) Comandos auxiliares e Funções de Conveniência

Aqui estão agrupadas as funções destinadas a auxiliar o usuário na utilização dos comandos dos grupos anteriores. São, geralmente, comandos destinados a imprimir caracteres . Note-se que não há diferença entre uma trajetória e um caracter , portanto, todos os comandos definidores de forma e de ação gráficas são válidos e realmente utilizados . Porém , como os caracteres são muito utilizados , criaram-se rotinas que já perfazem as operações necessárias de forma repetitiva , libertando o usuário .

Como exemplo:

TSHOW	{Imprime texto usando trajetórias}
BSHOW	{Imprime texto usando Bit-Map}
(texto)	{Define a mensagem a ser impressa}

Existem, também, comandos que permitem a extração de dados dos stencils e trajetórias para facilitar a programação de aplicações mais sofisticadas . Como exemplo:

GETCPT	{Retorna ponto corrente}
SETCPT	{Inicializa ponto corrente}
CURCPT	{Retorna ponto corrente do Stencil corrente}
VCPT, etc	{Retorna ponto corrente , no sistema virtual de coordenadas}

Definidos os quatro grupos , será iniciada a descrição da implementação das estruturas e comandos. Para isso serão utilizados exemplos , permitindo assim reconhecer o funcionamento de uma série de comandos auxiliares do núcleo básico da linguagem, já descritos.

VI-3)- Implementação dos STENCILS

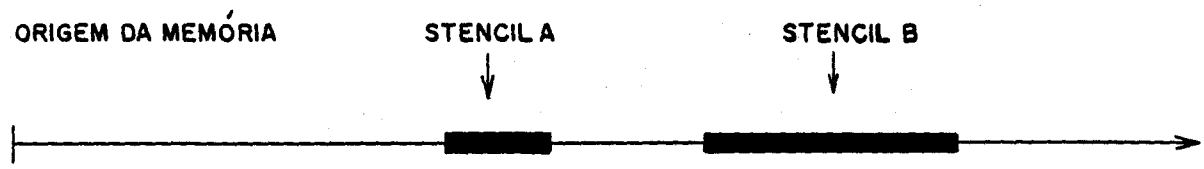
Como discutido no modelo gráfico, o Stencil é um conjunto bidimensional de pontos, que armazena as informações gráficas. Os Stencils podem ser utilizados de forma similar aos Stencils reais dos processos de fotocomposição, offset, matrizes fotográficas, etc (Warnock,82)

Como consequência do modelo gráfico , os stencils são as únicas entidades portadoras de imagens. Ou seja , a única forma de uma imagem ser visualizada, ou impressa, é através de um ou mais stencils. Para uma trajetória ser impressa é necessário que, primeiramente, esta trajetória seja colocada em um stencil, para que este seja direcionado ao papel.

Os stencils podem ter o tamanho que o usuário desejar . O único limite para o número e tamanho é o da memória disponível.

Fisicamente, os Stencils aqui implementados, constituem-se de uma região unidimensional de pontos, localizados na memória destinada a formação da imagem. O tamanho do Stencil é indiretamente determinado pelo número de pontos, ou memória, destinada a ele. Normalmente, a memória utilizada é constituida de memórias de vídeo , separada da memória destinada aos programas e dados normais de processamento . Tal fato advém da necessidade de haver uma rápida transferência destes dados para o vídeo ou dispositivo impressor, porém, ressalta-se que tanto o conceito, bem como as rotinas que implementam os stencils, são independentes deste fato. Tais rotinas são "device independents". A única entidade que carrega a informação do dispositivo, é a rotina que transfere os stencils prontos para o canal impressor , chamada SHOWPAGE.

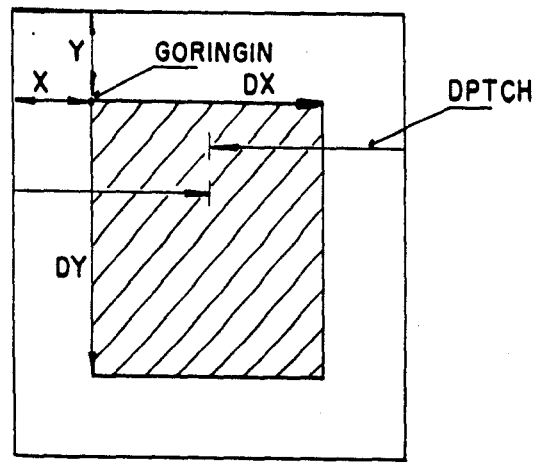
A memória de vídeo segue o esquema adotado pelo TMS34010, uma memória de bits ou pixels, apontados por um endereço linear. Os stencils seriam "pedaços" dessa memória:



Por conveniência, costuma-se representar a memória como um conjunto bidimensional ,retangular, aproveitando as facilidades do TMS34010 de trabalhar com formas, ou coordenadas, "XY" de endereçamento. Em outras palavras, um stencil pode ser considerado como uma tela , onde serão colocadas as informações usando as coordenadas XY . O sistema se encarrega, automaticamente, de converter estas coordenadas XY na posição linear da memória correspondente.

(Recomenda-se a leitura do capítulo 4, do TMS34010 User Guide), (Texas,86).

Dessa forma, um Stencil toma a aparência abaixo:



A origem dos eixos de coordenadas do Stencil é assumida , a priori, como sempre colocada no canto esquerdo superior, sendo este pixel de coordenadas 0,0. (E possível a especificação desta origem em qualquer um dos cantos do stencil, através da variável definidora de contexto VORG. Caso se deseje alterar, vide o comando VORG no apêndice e o manual do TMS34010).

Para a definição de um Stencil, usa-se o comando:

dy dx STENCIL nome

onde:

"dy","dx" são o número de pontos Y,X da dimensão desejado do Stencil.

"STENCIL" é um comando definidor de contexto;

"nome" é o nome do Stencil desejado, que será criado.

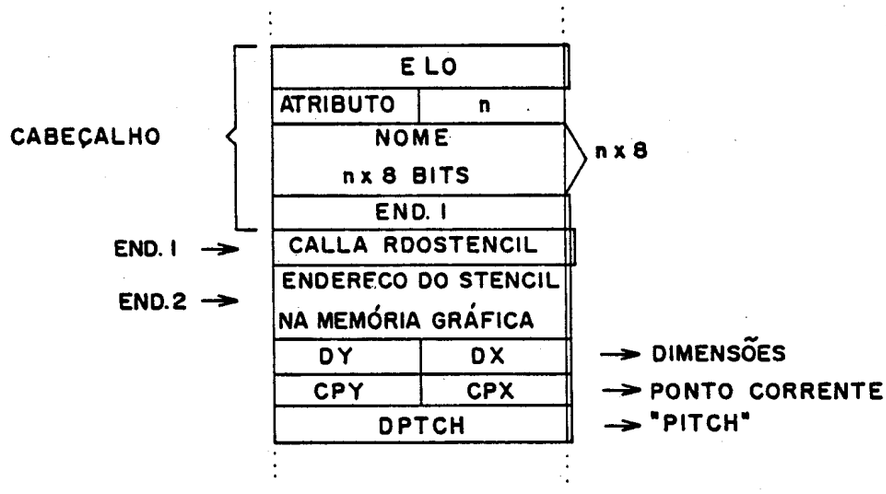
Por exemplo: O usuário deseja criar um stencil de nome PAG,

200 100 STENCIL PAG

No exemplo, cria-se o Stencil de nome PAG, com dimensão 200x100 pontos. Assim, a coordenada Y pode ir até 200 e a X até 100. Se em alguma função especificar-se uma coordenada deste stencil, que ultrapasse esses limites , o pixel acessado irá provavelmente atingir outro stencil. (O sistema , basicamente , não testa a violação de fronteira , porém este teste é perfeitamente possível de ser implantado pelo usuário , se desejar, através da modificação da função SETCPT).

Para definir-se um Stencil, é necessário posicioná-lo na memória gráfica. O comando STENCIL mantém um pointer chamado GORIGIN, que é incrementado pelo produto DY*DX a cada Stencil criado. Tal pointer vai se deslocando, de forma que o usuário não necessita monitorar o posicionamento na memória. Na forma que o núcleo está implantado , não é feita a verificação automática de estouro de memória, porém, o usuário pode criar tal controle, simplesmente verificando se GORIGIN ultrapassou o endereço limite do memória da implementação que está sendo usada.

Ao se criar um Stencil, é gerado no dicionário um cabeçalho, igual a todas as funções do MiniScript, na seguinte forma:



O "header" é o cabeçalho normal, utilizado em todas as funções da linguagem .

O endereço-1 é o endereço de chamada, ou execução da função. A rotina "RDOSTENCIL" perfaz o comportamento da função "STENCIL", que é o de simplesmente colocar na pilha o endereço-2.

O endereço-2 contém o endereço da origem do Stencil na memória de vídeo, destinada aos gráficos.

O par DYDX contém a dimensão Y-X, em número de pixels do Stencil.

O par CPY,CPX é uma variável auxiliar dupla , destinada a armazenagem do "CURRENT POINT" do Stencil, ou seja, contém o endereço do último ponto acessado no Stencil. É um pointer dentro do Stencil que será usado para diversas operações, descritas em breve.

O DPTCH é um parâmetro físico, calculado automaticamente por "STENCIL", que contém a diferença de endereços lineares de memória entre dois pixels de uma mesma coluna, separados por uma linha.

Este dado é usado na conversão das coordenadas YX para o endereço físico linear na memória gráfica. Normalmente DPTCH é igual a DX, mas o usuário pode criar sub-stencils, onde DPTCH é maior que DX. Outro fato relevante, é que o DPTCH deve levar em conta o tamanho do pixel. Como, normalmente, o pixel é de 1 bit, DPTCH = DX. Ao se invocar um Stencil, na pilha será colocado o endereço-2, no entanto, pode-se acessar as outras informações, como no exemplo:

```
PAG            {põe na pilha "endereço-2"}

PAG  @         {põe na pilha o endereço do Stencil na memória
                gráfica vídeo}

PAG 32 + @     {põe na pilha as dimensões DY DX}
                Obs.: 16LSB = DX , 16MSB = DY
```

É importante frisar que o cabeçalho do Stencil recém-criado, foi colocado no dicionário , no vocabulário contido em CURRENT. Ou seja , o cabeçalho está na memória de programa, contendo endereços que estão apontando para uma outra região de memória , situada na parte destinada à memória gráfica.

É possível, também, a criação de Stencils em posições desejadas pelo usuário, bem como a superposição deles na memória.

O comando DOSTENCIL permite a definição de Stencils especiais, através dos diversos parâmetros fornecidos a ele, todos fornecidos pelo usuário, como abaixo:

```
pitch   dy   dx   endereço   DOSTENCIL   nome
```

Neste caso:

"pitch" é a diferença de endereços de memória entre dois pontos de um coluna, separados por uma linha.

"dy" e "dx" são as dimensões Y,X do Stencil.

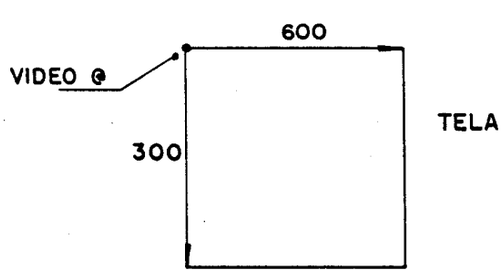
O "endereço", fornecido pelo usuário, indica a origem do Stencil na memória .
O "nome" é a denominação desejada.

Como exemplo, se o usuário deseja criar o Stencil PAG do exemplo anterior, seria:

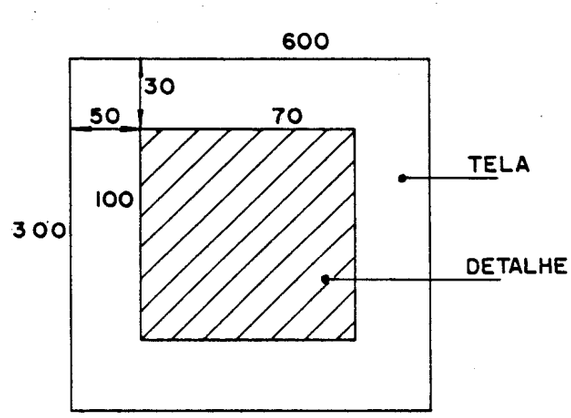
200	200	100	GORIGIN @	DOSTENCIL	PAG
pitch	dy	dx	endereço na memória gráfica	DOSTENCIL	nome

Agora, suponha-se que o usuário deseja criar um Stencil, chamado por ele de TELA, em uma posição da memória gráfica contida na variável VIDEO (também do usuário), tendo 600 pontos horizontais por 300 verticais. Para isto, o usuário deveria enviar os seguintes comandos:

600 300 600 VIDEO @ DOSTENCIL TELA



Suponha-se, novamente, que o usuário tem a necessidade de delimitar uma área incluída dentro do Stencil TELA, que será chamada DETALHE. Tal região teria todos os atributos de um Stencil , porém compartilharia uma certa região com TELA de forma coerente, ou seja, um Sub-Stencil.



Observando a figura, tem-se que:

pitch = 600, pois se esta dentro de TELA
dy = 100
dx = 70
endereço: endereço desejado de DETALHE, que deve ser calculado como endereço de TELA + endereço linear da origem de DETALHE dentro de TELA.

Na figura, seria calculado como:

```
VIDEO @  50  600  30  *  +  VIDEO2 !
```

A variável VIDEO2 , criada pelo usuário anteriomente, guarda o endereço de DETALHE, temporariamente.

Assim, a definição de DETALHE seria:

```
600   100  70   VIDEO2 @   DOSTENCIL   DETALHE
```

Observa-se que o "pitch" deve ser igual ao do stencil maior, TELA. Se o usuário adotasse um "pitch" diferente, não haveria concordância de imagem.

O stencil DETALHE possui todas as características comuns aos outros stencils , porém tem a propriedade de se situar "dentro" de outro ,chamado TELA , compartilhando certa região da memória de forma coerente, ou seja, a imagem de DETALHE seria um "detalhe" da imagem TELA. Este artifício permite uma grande agilização de processamento em algumas aplicações, onde detalhes devem ser retirados ou colocados com grande frequência. Assim, o detalhe passa a ter um nome e pointers independentes, não sendo necessário ficar calculando seus endereços e atributos a cada operação.

VI-4)- Operações com os STENCILS

Seguindo as conclusões decorrentes do Modelo Gráfico adotado, os Stencils podem ser conjugados de forma a simularem as operações gráficas reais da fotocomposição.

Assim, normalmente ocorre a seguinte situação:

Dois ou mais Stencils devem ser combinados de forma a produzir um terceiro. A um Stencil, é atribuido o status de ser o "source", ou fonte, e ao outro o "destination", ou tela do resultado. Entre esses dois stencils, ocorrerá uma ação de composição gráfica , aliada a uma operação propriamente dita, ou seja, podemos compor somando, compor através da operação AND, etc.

As ações de composição são uma indicação de quais Stencils serão processados e como. As operações são uma indicação de quais operações lógicas, ou aritméticas, que ocorrerão sobre as imagens, contidas nos Stencils. Temos 5 tipos de ações de composição, que na classificação, vista anteriormente, são considerados como definidores de ação gráfica, a saber:

COMPOSE

PASTE

DETACH

REPASTE

TRANSFER

a)- COMPOSE

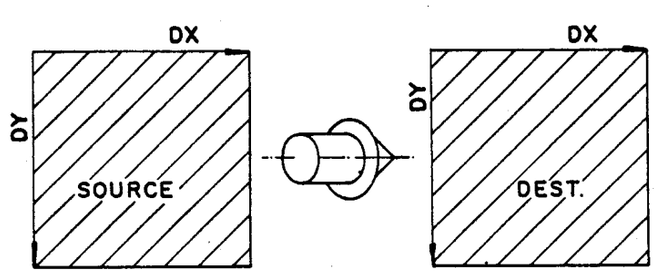
E a ação mais comum quando se tem dois Stencils , "Source" e "Destination", de mesmas dimensões, que serão sobrepostos. Com os definidores de contexto, já previamente selecionados, o usuário insere o comando:

source dest COMPOSE

Onde:

"source" é o nome do Stencil Fonte
"dest" é o nome do Stencil Destino, ou onde ficará o resultado.

Na figura abaixo, é mostrado um diagrama de como esta ação ocorre.



COMPOSE

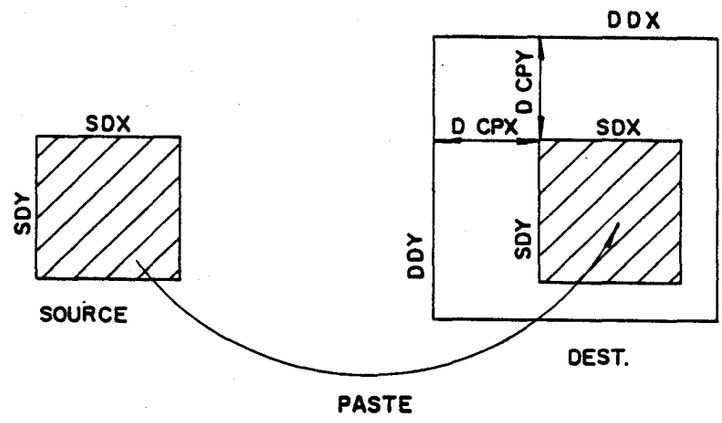
b)- PASTE

A ação PASTE coloca um Stencil "dentro" do outro.
O Source Stencil é colocado na posição corrente do Destination Stencil, ou seja, em CPY,CPX do Stencil destino. Após a ação, os pointers CPX,CPY do destino são deslocados da quantia CPY,CPX do Stencil Source. E a ação mais utilizada na composição de páginas, onde os "Source" Stencil são os "tipos" das letras e o "Destination" Stencil a "página". Note-se que podem ser de dimensões diferentes.

O comando pode ser introduzido da forma

source dest PASTE

Na figura abaixo, é diagramada a ação desta função

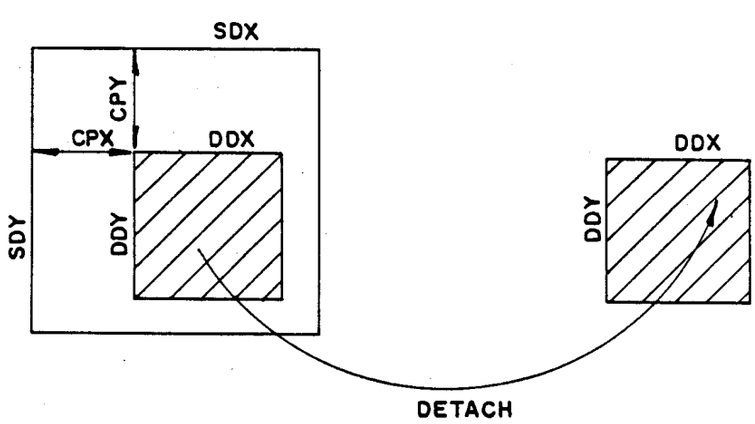


c)- DETACH

É a ação inversa do PASTE, sem alterar CPY,CPX, destinado a copiar detalhes dentro de um Stencil. Um stencil, com as especificações do "destino", é copiado de dentro do stencil "fonte" , apontado neste por CPY,CPX do fonte. Não é necessário que "destino" seja um sub-stencil de "fonte", pois, óbviamente, "destino" deve estar numa região diferente de memória. Não é necessário também que a região dentro de "fonte" , com as mesmas dimensões de "destino", seja um sub-stencil de "fonte".

O comando tem a forma:

fonte dest DETACH

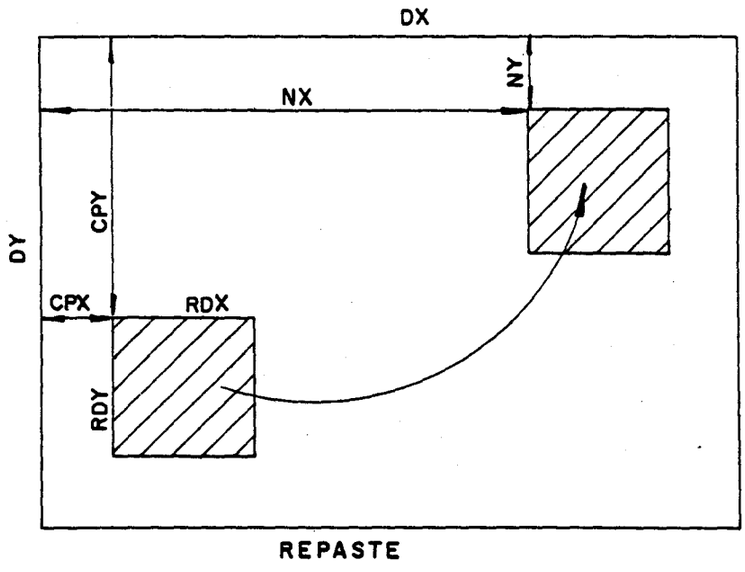


d)- REPASTE

E a mesma ação PASTE, operando dentro de um mesmo Stencil.
Serve para movimentar ou copiar regiões dentro de um Stencil maior.
O comando tem a forma:

source ny nx rdy rdx REPASTE

A ação é a de copiar um Substencil de dimensões rdy,rdx apontado por cpy,cpx em ny,nx. Não é necessário que estas regiões tenham sido definidas, anteriormente, como Sub-stencils. As regiões não devem se sobrepor , caso isto ocorra, o conveniente é eleger um stencil ponte.



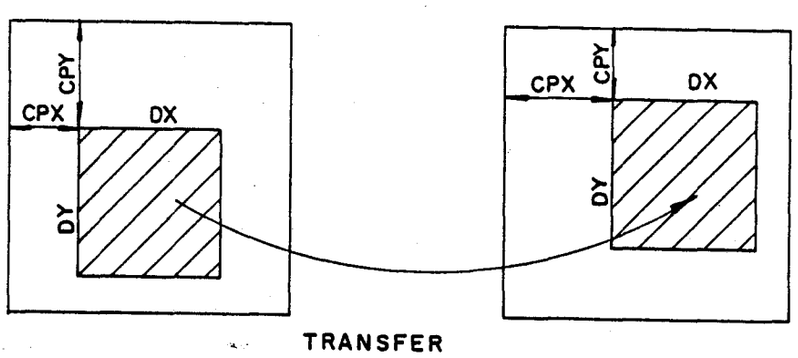
e)- TRANSFER

Copia o Stencil dy,dx, apontado por Source(CPX,CPY) em Destination(CPY,CPX).

Utilizado para transportar detalhes entre Stencils, porém não é necessário que sejam definidos como sub-stencils.

O comando tem a forma:

source dest dy dx TRANSFER



Durante uma ação de composição, é possível a ocorrência de operações lógicas ou aritméticas entre os elementos contidos no Stencil.

As operações a serem efetuadas são armazenadas como parâmetros, que serão utilizados nas ações gráficas futuras . Ou seja, as operações são determinadas nos definidores de contexto .

As operações agem tanto nas imagens contidas nos Stencils quanto nas trajetórias sendo aplicadas sobre estes stencils.

Estas operações são conhecidas como "Raster Operations" , destinadas especialmente a composição de imagems registradas em "bitmap", ou padrões "RASTER"(Newman,81)(Porter,84)(Pike,83).

Foram implementadas todas as "raster Op" disponíveis no TMS34010,que na implementação corrente, tem os seguintes nomes:

Nome	Operação
REPLACE	SOURCE -> DESTINATION
GAND	SOURCE AND DESTINATION -> DESTINATION
GANDNOT	SOURCE AND DESTINATION -> DESTINATION
BLACK	0 -> DESTINATION
GORNOT	S OR (-D) -> DESTINATION
GXNOR	S XNOR -> DESTINATION
GNOTD	(-D) -> DESTINATION
GNOR	S NOR D -> DESTINATION
GOR	S OR D -> DESTINATION
NOP	D -> DESTINATION
GXOR	S XOR D -> DESTINATION
GNOTAND	(-S) AND D -> DESTINATION
WHITE	! -> DESTINATION
GNOTOR	(S) OR D -> DESTINATION
GNAND	S NADN D -> DESTINATION
GNOTS	(-S) -> DESTINATION
GPLUS	S + D -> DESTINATION
GADDSTAT	ADD AND SATURATE S,D -> DESTINATION
GMINUS	D - S -> DESTINATION
GSUBSAT	SUBTRACT AND SATURATE S,D -> DESTINATION
GMAX	MAXIMUM S,D -> DESTINATION
GMIN	MINIMUM S,D -> DESTINATION

A operação REPLACE é a assumida pelo sistema na sua inicialização.

Uma vez selecionada a operação, esta é válida para todos as composições posteriores, até que nova operação seja escolhida.

Por exemplo:

REPLACE

LETRA PAG COMPOSE

A27 K4 PASTE

(obs: LETRA , PAG , A27 K4 são exemplos de stencils definidos pelo usuário)

Além da operações acima, existem as opções de transparência, onde todas as operações são processadas com a atributo de que quando o resultado der 0, tal zero funcionará como um "buraco" no Stencil. Esta função também é um definidor de contexto e para acioná-lo, basta se utilizar das contanstes TRUE e FALSE, como mostrado abaixo:

TRUE TRANSPARENCY (ativa)

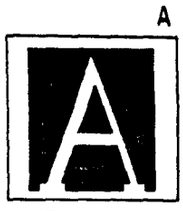
FALSE TRANSPARENCY (desativa)

Uma vez ativada a função, ela é válida para todas as ações gráficas posteriores, até que se deseje desativá-la.

Nos exemplos seguintes, serão alterados as operações desejadas, mantendo mesma ação gráfica para permitir uma melhor visualização dos recursos disponíveis.

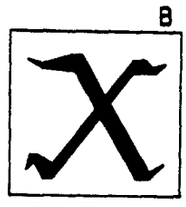
Por Exemplo, o usuário já definiu anteriormente os seguintes stencils que após uma série de processamentos, contém as seguintes imagens:

A



FONTE

B



DESTINO

BRANCO=1

Para especificar a operação desejada nas próximas ações gráficas, o usuário deve inserir o comando correspondente:

REPLACE

Não desejando a opção de transparência:

FALSE TRANSPARENCY

Para efetuar a ação desejada:

A B COMPOSE

O resultado é mostrado abaixo:

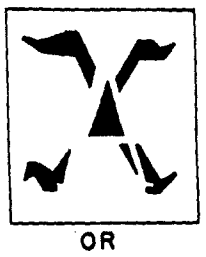
REPLACE

b) Supondo que o usuário, ao invés de REPLACE , usasse GOR:

GOR

A B COMPOSE

O resultado seria o seguinte:

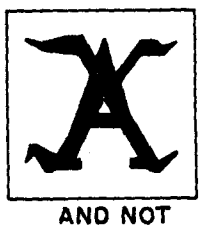


OR

c)- Se usasse GANDNOT:

GANDNOT

A B COMPOSE



AND NOT

d)- Se usasse AND:

GAND

A B COMPOSE

AND

e)- Se usasse GXOR:

GXOR

A B COMPOSE

XOR

VI-5) Algumas funções auxiliares para os STENCILS

São descritas, resumidamente, algumas funções de conveniência, que normalmente são utilizadas nas operações com stencils. No apêndice, são apresentadas algumas mais .

GETCPT

source GETCPT -> Y X
Pega o ponto corrente dentro do CPX,CPY do Stencil "source" e coloca na pilha. Normalmente utilizado para manipular ou registrar um ponto de interesse dentro de um stencil , antes que seja modificado por uma ação gráfica;

SETCPT

Y X Dest SETCPT
Atribui as coordenadas Y X como sendo o novo ponto corrente do stencil , CPX,CPY , ignorando o anterior. É a operação inversa de GETCPT;

CURSTENCIL

source CURSTENCIL !
Coloca o Stencil "source", como sendo o Stencil corrente para as trajetórias subsequentes. CURSTENCIL é uma variável de controle, que é acessada pelas trajetórias , indicando onde devem agir;

CURBRUSH

source CURBRUSH !
Coloca o stencil "source" ,como sendo o stencil que será usado como elemento simulador de "pincel".
CURBRUSH é uma variável de controle ,acessada pelas trajetórias quando devem traçar linhas, visando simular o efeito da espessura de um "pincel";

DOUBLEVAR

DOUBLEVAR nome
Cria uma variável dupla (Y,X), com a designação "nome". Ao inserir "nome" , este retorna seu endereço no topo da pilha;

D@

end D@
Double Fetch, recupera o par YX do endereço;

D!

y x end D!
Armazena o par YX no endereço;

ORIGIN

Variável dupla, que armazena o endereço que será usado como origem 0,0, para os comandos MOVETO, LINETO;
y x ORIGIN D! (armazena)
ORIGIN D@ -> y x (recupera)

CURCPT

CURCPT -> y x
Põe na pilha o ponto corrente do stencil corrente
Equivale a "CURSTENCIL @ GETCPT";

ADDXY y1 x1 y2 x2 ADDXY y3 x3
"double-add" , soma as coordenadas de dois pontos.

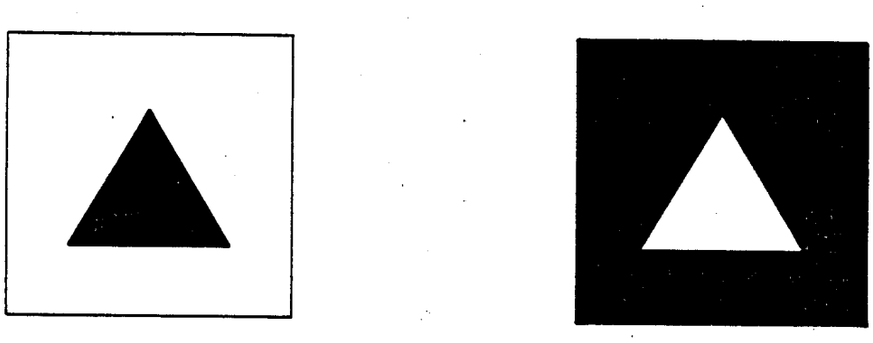
VI-6)- Implementação das trajetórias e dos comandos de pintura

O conceito de trajetória advém do fato prático de existirem figuras, símbolos, letras, que apesar de apresentarem diferentes "acabamentos", seguem uma "forma" comum.

Um exemplo bastante comum são as das letras. Uma letra, por exemplo o "A", pode ser grafada de várias formas:

Nos três exemplos, a trajetória que um hipotético "pincel", ou de um "lapis", são as mesmas, mudando sómente o "tipo" de pincel.

As trajetórias podem, também, servir para delimitar regiões que serão "pintadas", como por exemplo:



As trajetórias são uma sequência de rumos e deslocamentos, que devem ser repetidos pelo elemento "pintante". Dessa forma, uma trajetória nada desenha, dependendo, assim, de uma outra entidade que realmente marca ou "coloca a tinta", seguindo o caminho pré-estabelecido.

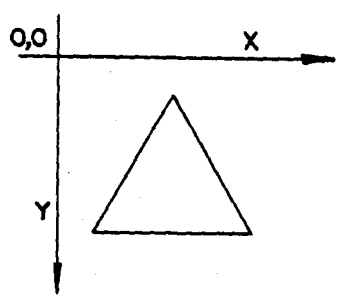
Os rumos ou caminhos definidos nas trajetórias podem se utilizar de sistemas de coordenadas absolutas ou relativas.

As absolutas definem os pontos, com a origem do sistema de coordenadas da trajetória coincidindo com a origem do sistema do stencil corrente. As relativas se utilizam de um ponto, tomado como origem 0,0 , situado no ponto X,Y do stencil corrente, ponto este armazenado na variável dupla ORIGIN, previamente. Existe, também, a possibilidade de se criar trajetórias com deslocamentos incrementais.

Uma trajetória é conhecida pelo seu nome, especificado durante sua criação. Uma trajetória , na estrutura da linguagem, nada mais é do que uma função, construída a partir da união coerente dos comandos definidores de forma.

Por exemplo, o usuário deseja criar a trajetória chamada TRIANGULO, utilizando-se de coordenadas relativas .

```
: TRIANGULO  20   60   MOVETO   100   100   LINETO
             100  20  LINETO    60    20   LINETO  ;
```



Após a inserção, TRIANGULO foi definido no dicionário como uma função, que representa a forma de triângulo da figura.Tal figura, do ponto de vista da estrutura da linguagem ,nada difere das outras funções, possuindo o cabeçalho, chamadas para outras funções, operações matemáticas, etc.

A principal diferença é que contém uma série de chamadas a outras funções gráficas, definidoras de forma.

O comando "20 60 MOVETO" serve para posicionar o "pincel" na posição Y=20, X=60, sem riscar. Tais posições, no caso do comando MOVETO, são coordenadas relativas a origem 0,0. Tal origem, neste caso, representa um imaginário sistema de coordenados, ou "Virtual Coordinate System", que pode ser posicionado em qualquer ponto do Stencil, onde a trajetória será executada, através do comando ORIGIN.

O comando "100 100 LINETO" serve para deslocar o "pincel" até a posição Y=100, X=100, neste caso, "riscando" a partir da última posição, Y=20, X=60.

Os comandos "100 20 LINETO" e "60 20 LINETO" completam o triângulo desejado. Os comando "LINETO" caminham numa linha reta e o algoritmo usado para este fim é o conhecido algoritmo de BRESENHAM (Bresenham,65), (Newman,81).

Até agora, pode-se observar que somente foi "compilada" uma trajetória , nenhum gráfico ou desenho foi efetivamente feito.

As trajetórias podem ser utilizadas para serem os caminhos que os "pinceis" irão usar, ou os caminhos que delimitam uma região que será pintada.

Quando a trajetória for usada para "pincelar", ou "riscar", deverá o sistema estar no modo chamado DRAWMODE. Neste caso um Stencil é usado para simular um pincel, ou "BRUSH", para criar os padrões de linhas grossas, traçados especiais , etc.

Quando a trajetória for usada para delimitar certa região, o sistema deverá estar no FILLMODE.

VI-7)- Trajetórias no DRAWMODE

O DRAWMODE é um estado no modo gráfico onde as trajetórias serão usadas para traçarem linhas, que realmente serão colocadas no stencil.

No DRAWMODE a trajetória usa o pincel adotado de antemão pelo comando CURBRUSH. Para se entrar neste modo de operação , basta o usuário inserir o comando, como abaixo:

DRAWMODE

Como exemplo,suponha-se que o usuário deseja desenhar, no Stencil PAG7, a trajetória TRIANGULO, usando o pincel K5, a partir do ponto corrente em PAG7. Tanto o stencil PAG7, quanto o pincel K5 foram criados anteriormente pelo usuário. O usuário também já adotou como operação padrão o comando (definidor de contexto) REPLACE .

Primeiramente deve-se nomear o Stencil corrente, ou destino.

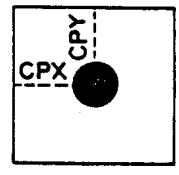
PAG7 CURSTENCIL !

A partir daqui, todas as trajetórias serão executadas sobre o Stencil PAG7.

O pincel adotado é o K5, como na figura abaixo.

K5 CURBRUSH !

A partir daqui , todos os comandos MOVETO,LINETO usarão o "pincel" K5



K5

Para simular o efeito do pincel sendo "escorregado", e deixando a "tinta", é necessário adotar a função GOR (função lógica OR gráfico).

GOR

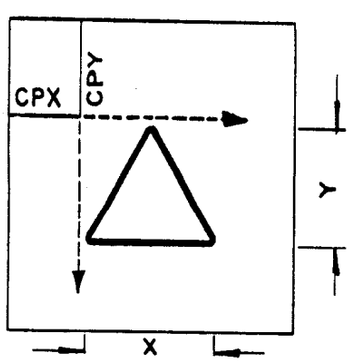
A origem da trajetória foi escolhida pelo usuário como sendo o ponto corrente de PAG7, portanto, o usuário deverá obter o ponto corrente do stencil e inicializar a variável ORIGIN.

PAG7 GETCPT ORIGIN D!

Para a execução, basta inserir o nome da trajetória desejada:

TRIANGULO

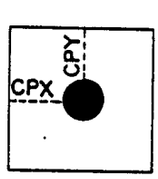
O resultado no PAG7 será o seguinte:



Observe que o pincel K5 é um Stencil, que será impresso em cada ponto da trajetória, em sucessivas superposições, criando a ilusão de um "pincel" sendo "escorregado".

Todas as regras e comandos válidos para os Stencils são válidos para os pinceis, incluindo as operações.

A diferença principal entre um Stencil normal e um que será usado como BRUSH, é a necessidade de que o ponto corrente no BRUSH esteja no centro da figura que representa a "impressão" do pincel. No exemplo, K5:



K5

Como as trajetórias são referenciadas a um sistema de coordenadas imaginário, é possível que este seja rotacionado e expandido antes de ser aplicado ao Stencil corrente.

Através dos comandos definidores de contexto SCALEX, SCALEY, SHEARX, SHEARY , ROTATION , etc, é possível determinar quais as transformações de coordenadas as trajetórias vão sofrer quando executadas.

Por exemplo , o usuário deseja produzir uma ampliação na direção Y , para isto deve inserir:

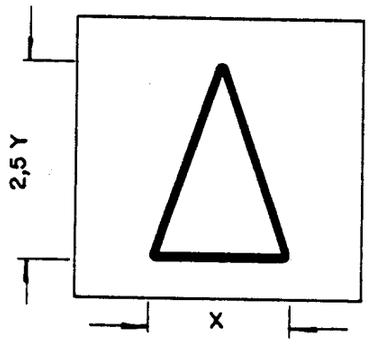
5 2 SCALEY D!

Observa-se que até aqui , nada foi alterado na trajetória. Porém ao usuário executar , inserindo:

TRIANGULO

multiplicará, no exemplo, todas as coordenadas Y por 5/2 ou 2,5.

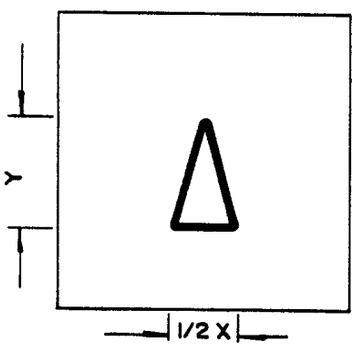
Assim, no Stencil PAG7 o usuário obterá:



O mesmo poderia ter ocorrido no eixo X, através de:

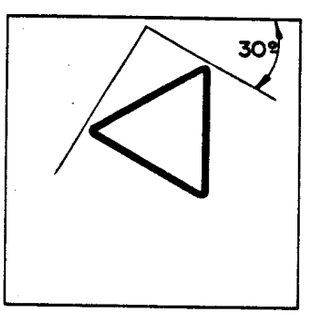
```
1   2   SCALEX   DI
TRIANGULO
```

Onde, neste exemplo, o usuário estaria multiplicando X por 1/2.



A rotação dos eixos é obtida através de:

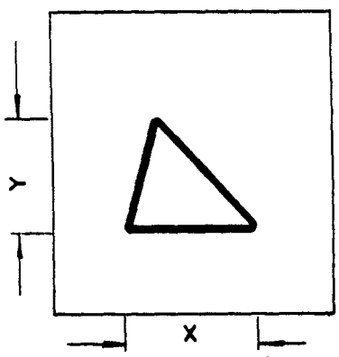
```
-30      ROTATION
TRIANGULO
```



O comando ROTATION promove a rotação no sentido anti-horário, pelo número de graus fornecido, no caso -30 (horário).

Outra transformação possível seria o SHEAR, especialmente útil na confecção de caracteres itálicos.

```
5     3     SHEARX D!

TRIANGULO
```



Tal expressão equivale a transformar todas as coordenadas X em X+(5/3)Y .

As funções SCALE, ROTATION e SHEAR são acumulativas e válidas para todas as trajetórias a serem executadas a seguir, até que novos parâmetros sejam dados a tais definidores de contexto .

As trajetórias possuem propriedades associativas. É possível unir ou compor trajetórias para formar uma maior.

Por exemplo:

```
: ESTRELA     TRIANGULO1     TRIANGULO2 ;
```

Aqui o usuário criou a trajetória ESTRELA , produto da união das trajetórias definidas anteriormente como TRIANGULO1 e TRIANGULO2.

```
: TRIAN     TRIANGULO     0     0     MOVETO ;
```

Neste exemplo, a trajetória TRIAN é composta da trajetória TRIANGULO somada ao comando 0 0 MOVETO.

VI-8)- Trajetórias no FILLMODE

No FILLMODE, as trajetórias são utilizadas para delimitar regiões que serão pintadas ou preenchidas. Neste caso, não se usa mais o pincel ou CURBRUSH, simplesmente se marca a trajetória , de forma a delimitar regiões que formarão as figuras a serem "enchidas de tinta".

Para se determinar este modo , basta o usuário inserir

FILLMODE

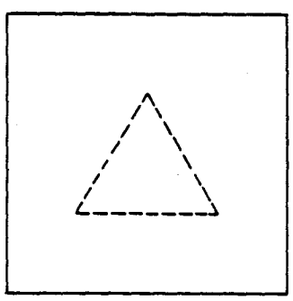
Tudo o que foi discutido para as trajetórias no DRAWMODE , com exceção do "pincel" ,continua válido .

Por exemplo, se o usuário executar a seguinte sequência no FILLMODE:

PAG7 GETCPT ORIGIN DI

TRIANGULO

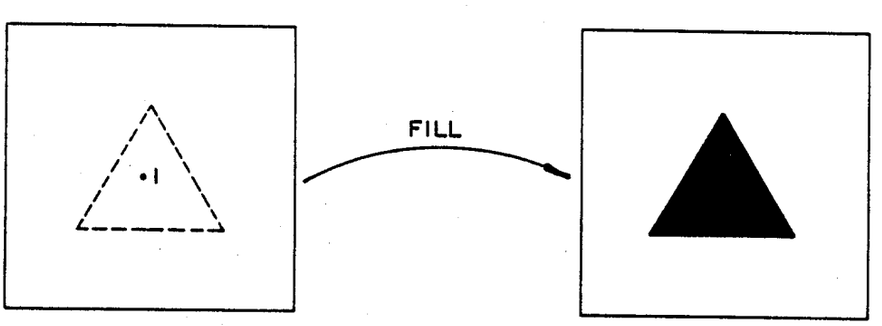
A trajetória TRIANGULO será desenhada sobre o Stencil PAG7. Observando o PAG7, nota-se que o TRIANGULO foi marcado com um pincel imaginário, de espessura de 1 ponto, com o valor definido previamente pela variável BOUNDARY.



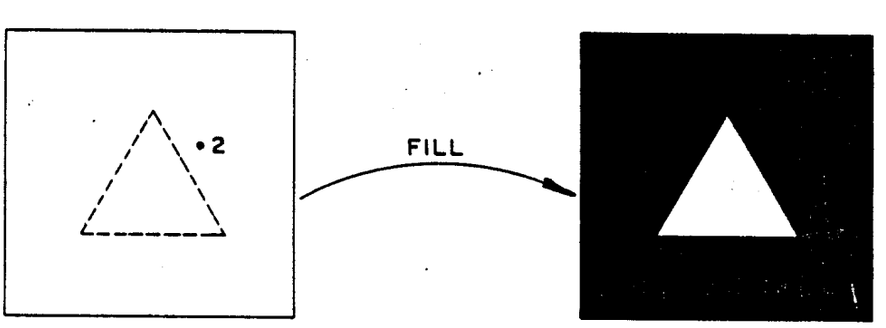
Se BOUNDARY for TRUE, o TRIANGULO será riscado com um pincel de espessura 1, de valor 1, ou seja, "PRETO". Se BOUNDARY for FALSE, o pincel será de espessura 1, mas de valor 0, ou "BRANCO". (Tal dualidade visa a permitir riscar sobre fundos pretos ou brancos).

Para se pintar o TRIANGULO, o usuário deve agora se utilizar do comando FILL. Tal comando pega um ou mais pontos na pilha e os usa como "sementes" para espalhar a tinta sobre o objeto. Tais pontos devem ser apontados para regiões internas a figura, onde se deseja pintar. A "tinta" utilizada é a mesma que foi utilizada em BOUNDARY (ou seja BRANCO ou PRETO).

Se no Stencil PAG7 o usuário escolher o ponto 1, interno ao TRIANGULO, obterá o seguinte:



Se escolher o ponto 2, externo ao triângulo, obterá:



A função FILL é uma operação recursiva que explora os pontos adjacentes às "sementes" fornecidas , verificando se as bordas da figura (delimitado por BOUNDARY) são atingidas. Caso não sejam , pinta os pontos adjacentes com o valor de BOUNDARY e verifica os próximos pontos. O algorítmo utilizado é conhecido como "scan line seed fill" (Rogers, 85), (Smith,79). Tal algorítmo trabalha diretamente sobre o stencil , varrendo as linhas "horizontais" e preenchendo-as de "tinta" seguindo aproximadamente os passos descritos a seguir:

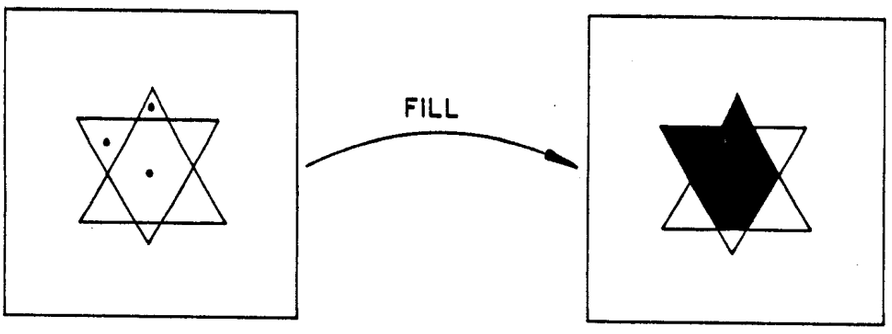
- a coordenada de um ponto dentro da região a ser preenchida, é colocada na pilha pelo usuário, logo a seguir de um "flag" indicativo de "fim de pilha" . Este ponto arbitrário dentro da figura é a semente inicial;,
- a seguir, o algorítmo pega da pilha a "semente" , "pinta" este ponto dentro da figura e vai pintando todos os pontos a esquerda e a direita, até que a borda seja atingida;
- o algorítmo registra a borda esquerda como "xleft" e a direita como "xright";
- no campo xleft <= x <= xright, o algorítmo verifica a linha "y+1" (linha imediatamente abaixo), buscando a existência de algum ponto ainda não pintado . Caso haja pontos não pintados, o algorítmo busca o ponto mais a direita e coloca sua coordenada na pilha;
- o algorítmo repete o passo anterior para a linha "y-1" (linha imediatamente acima), colocando também na pilha o ponto não pintado situado mais a direita ,
- a seguir , verifica se a pilha está vazia (testando se o topo da pilha contém o "flag" indicativo). Se houver pontos , reinicia o processo e caso contrário, termina;

Este algorítmo possue uma série de variantes, que o leitor poderá verificar nas referências (Pavlidis,79), (Pavlidis,81). Por ser um algorítmo recursivo, e se utilizando da pilha , foi relativamente simples sua implementação no sistema. Por ser um algorítmo que coloca "linhas adjacentes" em análise , sua utilização da pilha de parâmetros e da pilha de retorno não é muito grande.

O algorítmo é capaz de preencher qualquer região "fechada" , mesmo que contenha "buracos", como num anel.

Os pontos "sementes" podem ser fornecidos em qualquer número. O sistema vai buscando, na pilha, até encontrar o flag TRUE (-1). Porém , para cada região "fechada" , somente um ponto é necessário. Seguindo o raciocínio ,muitos pontos "sementes" são necessários quando se deseja preencher muitas regiões fechadas , não conectadas. Tal procedimento permite, também, que em figuras complexas o usuário decida quais regiões serão "pintadas", ou não.

Por exemplo:



Caso a semente esteja fora de um contorno fechado, ou caso o contorno seja "aberto" ou "furado", a "tinta" "vaza", inundando todo o Stencil.

Voltando ao exemplo do Stencil PAG7, para que o usuário comande a operação de pintura , deve introduzir:

TRUE 750 500 FILL

As coordenadas 750 500, relativas ao sistema de coordenadas do Stencil, são os pontos YX interno ao TRIANGULO. Deve-se lembrar, sempre, que um ponto tem "dois números", representando as coordenadas y ,x . Tais pontos estão sempre referenciados ao sistema de coordenadas do stencil e não ao da trajetória.

Para evitar que em cada posição que a trajetória TRIANGULO seja desenhada, o usuário tenha que determinar manualmente um ponto interno em coordenadas do Stencil, basta definir a trajetória, de forma que o último ponto seja posicionado dentro da figura e a seguir, pelo comando GETCPT, a "semente" vá para a pilha.

Para isso, o usuário pode , por exemplo, definir:

```
: TRIANGFILL TRIANGULO 75 60 MOVETO ;
```

Ao se executar TRIANGFILL, o último ponto da trajetória será convertido por ORIGIN + (75,60) e será o ponto corrente do Stencil PAG7. Dessa forma, introduzindo:

```
TRIANGFILL

TRUE    PAG7    GETCPT    FILL
```

será o suficiente para desenhar o TRIANGULO e colocar a semente para FILL.

Os atributos SCALE, ROTATION, SHEAR são válidos no FILLMODE, bem como todas as características das trajetórias discutidas no DRAWMODE.

VI-9)- Outros comandos definidores de forma e auxiliares para as trajetórias

Existem uma série de comandos visando auxiliar o usuario no trabalho com as trajetórias. São apresentados, a seguir, alguns destes comandos ,sendo que no apêndice está a relação completa.

AMOVETO — y x AMOVETO
Movimento, em coordenadas absolutas do stencil. Atualiza o ponto corrente do stencil corrente para y,x. Os parâmetros SCALEX,Y ROTATION , SHEARX,Y não são utilizados.

ALINETO — y x ALINETO
Traça uma linha, partindo do ponto corrente do stencil até y,x , em coordenadas absolutas. Os parâmetros SCALEX,Y , ROTATION , SHEARX,Y não são utilizados.

RMOVETO — dy dx RMOVETO
Movimento em coordenadas relativas , em valores incrementais. Atualiza o ponto corrente do stencil somando os incrementos . Os parâmetros SCALEX,Y ROTATION e SHEARX,Y são válidos.

RLINETO — dy dx RLINETO
Traça uma linha, a partir do ponto corrente seguindo os incrementos dy,dx (movimentos relativos) . Os parâmetros SCALEX,Y , SHEARX,Y e ROTATION são válidos.

VCPT — VCPT D@ -> y x
Variável dupla, que contém o último ponto fornecido a um comando MOVETO ou LINETO.

VI-10) Implementação dos caracteres

Os caracteres, no modelo gráfico adotado, não são uma entidade com características distintas dos Stencils ou trajetórias.

Os caracteres podem ser definidos como um conjunto de Stencils, ou como um conjunto de trajetórias, de acordo com a necessidade e a conveniência do usuário.

Um conjunto de caracteres, definido como Stencil, tem a vantagem de ter um processamento mais rápido, pois dada a letra desejada a ser impressa, basta copiar o Stencil "letra" para a posição desejada no Stencil "página". É um processo similar a impressão de caracteres em memórias de vídeo. A desvantagem é a de ocupar maior espaço na memória, pois é necessário existir a tabela dos caracteres e serem usados já em forma de Stencils.

Como existe uma grande variedade de tipos de caracteres , o acúmulo de todos os stencils, dos diversos tipos, tornaria a necessidade de memória proibitiva. Outra desvantagem, é a de não permitir transformações, tais como: rotações, ampliações, etc, de forma rápida, sem a introdução de erros. Existem alguns algorítmos para este tipo de operação , incluindo operações de minimização de erros , porém no sistema básico, aqui apresentado, não foram implementados (Crow,78), (Casey,82).

Os caracteres, definidos como trajetória, tem a vantagem de permitir todas as tranformações desejadas, ampliações, rotações em eixos, composições, etc e ocupa pouquíssimo espaço na memória. A desvantagem é que,toda vez que um caracter é solicitado, devemos desenhar a trajetória na posição desejada, seguindo as tranformações especificadas e depois "pintá-los". Tais operações consomem mais tempo.

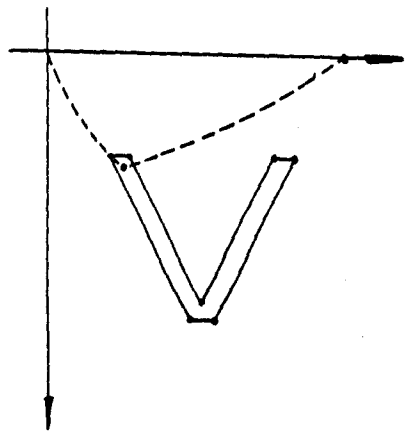
Uma solução intermediária é a de armazenar os caracteres como trajetórias e usá-los como Stencils, da seguinte forma.

Dado o texto a ser impresso, as escalas e rotações adotadas, etc, constrói-se uma tabela com os Stencils desses caracteres a serem usados , a partir de suas trajetórias. Note que, em um texto, as letras utilizadas se repetem muitas vezes, não sendo necessário, portanto, seu re-cálculo, pois já existe um "Stencil" criado pela mesma letra usada antes. Através deste esquema, a utilização de memória é media, com uma velocidade maior que se usassemos somente trajetórias. A desvantagem é a de obrigar o usuário "preparar" o "menu" de caracteres de antemão.

Para que a filosofia da implementação dos caracteres seja compreendida , será utilizado um exemplo .

Suponha-se, agora, que o usuário deseja definir a letra "V" como uma trajetória.

Na figura seguinte,é apresentado o esboço da trajetória, segundo a concepção artística do usuário:



Segundo o esboço , pode-se codificá-lo da seguinte forma:

```
: LETRAV   10 15 MOVETO   10 20 LINETO   40 30 LINETO
           10 40 LINETO   10 45 LINETO   50 35 LINETO
           50 25 LINETO   10 15 LINETO
           20 20 MOVETO   CURCPT
           0  55 MOVETO
;
```

Da forma que a letra "V" acima foi definida, observa-se que os comandos "20 20 MOVETO CURCPT" colocam na pilha o ponto ORIGIN+20,20, que será utilizado como semente para FILL.

O comando "0 55 MOVETO" posiciona o ponto corrente, de forma que o próximo caracter não seja sobreposto ao primeiro.

Assim, como o usuário definiu sua letra "V", pode definir também uma série de caracteres que formam um conjunto, seguindo o mesmo estilo "artístico". Tal conjunto, o usuário deseja chamar aqui de TESTEFONT e terá toda a tabela ASCII, portanto 126 caracteres.

Cada letra usada terá uma posição de memória e deverá ser "administrada" por um elemento, que centralizará tal conjunto.

Para isso , o usuário definirá o tal elemento centralizador:

```
126   ARRAY   TESTEFONT
```

A letra "V" tem como código ASCII, o número 86 (decimal), assim:

```
' LETRAV  86  TESTEFONT !
```

Explanando o ocorrido, observa-se que pelo comando ARRAY, o usuário criou uma matriz unidimensional de 126 posições, batizada de TESTEFONT. A seguir , através do comando ' (tick) LETRAV , o endereço de execução da trajetória LETRAV foi colocada no topo da pilha , que será armazenado na posição 86 da matriz TESTEFONT.

Seguindo o mesmo raciocinio , o usuário criou as trajetórias das outras letras e , uma a uma ,foi colocando seus respectivos endereços de execução nas posições ,códigos ASCII, correspondentes ao caracter . Tem-se, finalmente, que a matriz TESTEFONT armazena todos os endereços das trajetórias, que representam as letras, que compõe o "type-font" TESTEFONT.

A matriz TESTEFONT é o "encoding vector", ou vetor de codificação do "conjunto tipográfico" , ou "fonte" do usuário.

Caso o usuário deseje agora executar a trajetória da letra "V", do conjunto tipográfico "TESTFONT" ,no stencil PAG7 , já definido anteriormente , basta:

```
PAG7 CURSTENCIL !

86 TESTEFONT @  EXECUTE
```

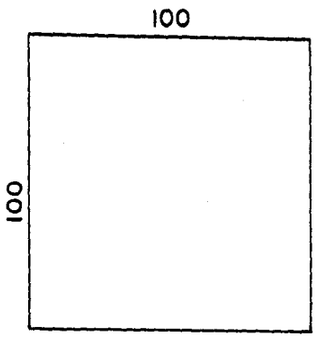
Para cada "fonte" existe o vetor de codificação correspondente.

Selecionando o "encoding vector", automaticamente se seleciona o conjunto de caracteres que perfazem o "fonte".

O fonte TESTEFONT armazena um conjunto de caracteres definidos por trajetórias. Agora, o usuário deseja definir outro fonte, que ao invés de trajetórias, será composto por Stencils. Novamente, o usuário vai definir a letra "V" , usando a trajetória já definida LETRAV em um Stencil. Primeiramente se cria o stencil:

```
100 100 STENCIL  V-BITMAP
```

O nome da letra V, neste fonte, será V-BITMAP e tem a dimensão de 100x100 pontos.



Será aproveitada a definição da letra "V", em trajetória do fonte TESTFONT, desenhando-se no stencil recém criado V-BITMAP.

Primeiramente, o usuario especifica a função desejada e o modo.

```
REPLACE

FILLMODE
```

Seleciona o Stencil:

```
V-BITMAP   CURSTENCIL !
```

Seleciona a transformação e escalas desejadas.

```
1     1     SCALEX  D!

1     1     SCALEY  D!

0           ROTATION
```

Seleciona o ponto onde a letra será desenhada (o current point inicial do V-BITMAP).

```
0 0 AMOVETO
```

e armazena a origem:

```
CURCPT  ORIGIN  D!
```

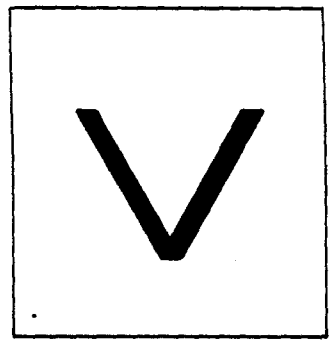
Finalmente, o usuário "desenha" e "pinta" a letra,

```
TRUE  LETRAV  FILL
```

ou usando o "encoding vector":

```
TRUE   86 TESTFONT @ EXECUTE    FILL
```

o resultado será:



Analogamente, o usuário define novamente um outro vetor de codificação,

```
126  ARRAY  BITMAPFONT
```

e direciona a letra V na sua posição correspondente.

```
' V-BITMAP  86  BITMAPFONT !
```

Repetindo o mesmo procedimento para outras letras , o usuário montou o conjunto tipográfico ,"fonte", de nome BITMAPFONT, que agrupa o conjunto de stencils com as letras desejadas.

O usuário possui agora dois "conjuntos tipográficos", que serão utilizados na impressão dos textos desejados.

VI-11)- Imprimindo o texto

O texto, a ser impresso, deve ser fornecido da seguinte forma: entre parenteses, "(" e ")".

(Texto a ser impresso)

Tudo o que estiver entre os parenteses, será reconhecido como caracteres a serem impressos, descontando os espaços em branco imediatamente antes e após os parenteses, que são obrigatórios.

Suponha-se que o usuário tem a seguinte mensagem a ser impressa, denominada por ele de "TEXTOTESTE", desejando imprimir as letras "VV":

: TEXTOTESTE (VV) ;

O usuário vai imprimir tal mensagem no Stencil PAG7.

PAG7 CURSTENCIL!

Na posição 10 10:

10 10 AMOVETO

A operação será GOR, para que o quadro de uma letra não apague partes da letra anterior.

GOR

Para selecionar o "fonte" desejado, neste exemplo o Fonte BITMAPFONT, de Stencils:

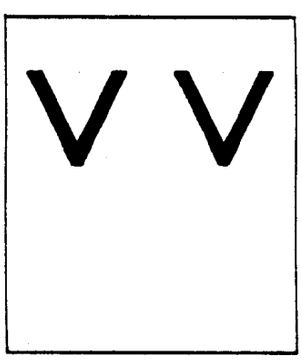
SELECTBMFONT BITMAPFONT

O comando SELECTBMFONT especifica o Fonte de caracteres em Stencils, que será usado futuramente para a impressão de texto. Este comando armazena numa variável de controle interna chamada CURBMFONT, o nome (endereço) do "fonte" escolhido.

Para a impressão:

TEXTOTESTE BMSHOW

O resultado , no stencil PAG7:



O comando BMSHOW imprime o texto, usando o Fonte corrente (armazenado em CURBMFONT), no Stencil corrente (armazenado em CURSTENCIL).

Se o usuário desejasse usar o "fonte" TESTEFONT, de trajetórias:

SELECTTFONT TESTEFONT

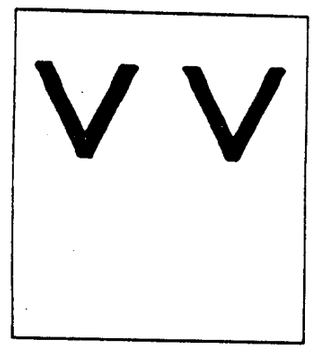
Analogamente , o comando SELECTTFONT especifica o Fonte de caracteres codificados em trajetórias, que será usado por TSHOW, colocando na variável CURTFONT seu nome. O comando TSHOW show imprime o texto desejado , usando o "fonte" de trajetórias armazenado em CURTFONT no stencil, contido em CURSTENCIL.

Como se está usando trajetórias , o modo selecionado pode alterar o desenho resultante.

Para a impressão o usuário comanda:

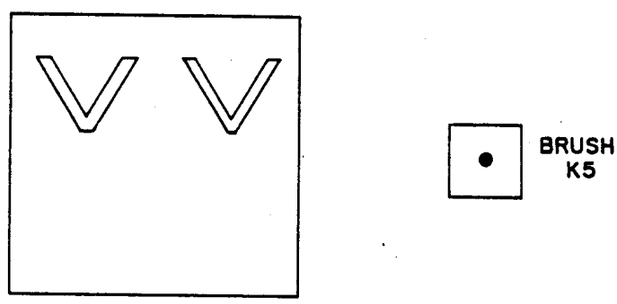
TEXTOTESTE TSHOW

Se estiver no FILLMODE, o usuário obterá:



Se estiver em DRAWMODE, o CURBRUSH será utilizado como pincel.

Supondo que CURBRUSH seja o "pincel" K5 dos exemplos anteriores, o Stencil PAG7 terá:



O comando TSHOW imprime o texto, usando o Fonte corrente de trajetórias (armazenado em CURTFONT), efetuando às transformações correntes (SCALE, SHEAR, ROTATION) e colocando, através da operação gráfica especificada de antemão, GOR, o texto desejado no stencil corrente PAG7.

VI-12)- Alguns exemplos usando as estruturas da linguagem

Serão apresentados, a seguir, alguns exemplos de utilização. Novas estruturas e conceitos são apresentados.

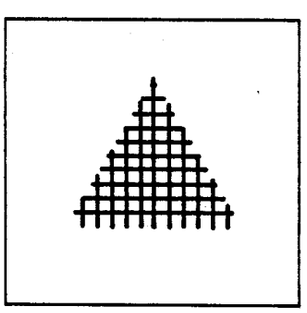
Os exemplos foram testados na implementação física, descrita no capítulo a seguir.

Os Stencils usados, geralmente tem o "pitch" do monitor gráfico utilizado, visando não distorcer a "imagem" apresentada ao usuário.

As palavras colocadas entre chaves { , } não são consideradas como comandos pelo sistema e sim comentários , permitindo a colocação de notas explicativas.

a)- Exemplo 1:

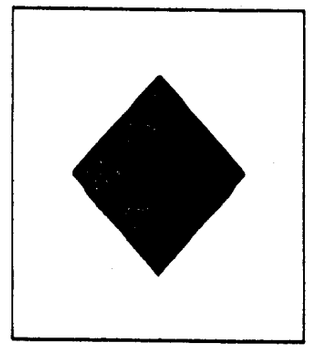
O usuário deseja obter o seguinte desenho:



A solução usada pelo usuário:

```
GRAPHICS  FALSE  TRANSPARENCY  TRUE BONDARY  !     (Seleciona  o
vocabulário gráfico e os atributos desejados)

4096  200  600  0  DOSTENCIL  TELA      (Define o Stencil TELA)


(O "pitch" adotado é devido à tela do monitor gráfico utilizado)


4096   150   300  84720  DOSTENCIL  MASCARA   (Define o  Stencil
MASCARA)

4096  10  40  2800  DOSTENCIL  STROKE3  (Define o Pincel)

BLACK  TELA  TELA  COMPOSE    (Limpa o Stencil TELA)

MASCARA  MASCARA  COMPOSE     (Limpa o Stencil MASCARA)

STROKE3  STROKE3  COMPOSE     (Limpa o Stencil STROKE3)


(Define a trajetória que desenha as linhas horizontais)

: LINHAHOR  11  0  DO

            I  10  *  DUP  0  SWAP  250

            MOVETO  LINETO  LOOP;


(Define  a trajetória que desenha as linhas verticais)

: LINHAVER  11  0  DO

            I  30  *  DUP  0  SWAP  100

            SWAP  MOVETO  SWAP  LINETO  LOOP  ;


(Define o Pincel)

FILLMODE  REPLACE  STROKE3  CURSTENCIL  !

5  15  AMOVETO  3  20  ALINETO  5  25  ALINETO

7  20  ALINETO  5  15  ALINETO  5  20  ALINETO

TRUE  CURCPT  FILL
```

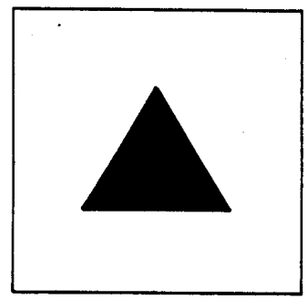
No Stencil STROKE3 (ampliado):



(Define a Máscara)

```
MASCARA  CURSTENCIL  !

15   150  AMOVETO  115  225  ALINETO  115  75  ALINETO   15   150
ALINETO  TRUE  65  150  FILL
```

No Stencil MASCARA:



(Define o reticulado)

DRAWMODE TELA CURSTENCIL ! GOR (Especifica o Stencil e a operação)

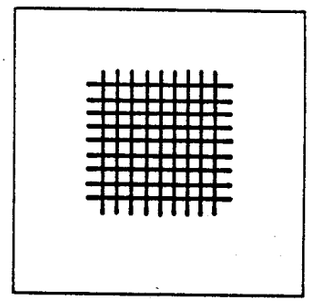
STROKE3 CURBRUSH ! (Especifica o pincel)

1 1 SCALEX D! 1 1 SCALEY D! (Tamanhos)

35 50 AMOVETO CURCPT ORIGIN D! (Posiciona)

LINHAHOR LINHAVERT (Desenha as trajetorias)

Em TELA ter-se-á:

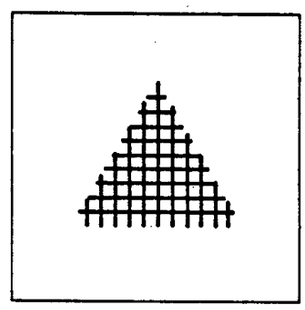


(Compõe, usando a função gráfica "AND")

GAND 20 20 AMOVETO (Especifica operação e posição)

MASCARA TELA PASTE

Em TELA obter-se-á:



Na foto abaixo, tem-se o exemplo, como apresentado na tela, de um monitor gráfico. Observa-se a MASCARA e o STROKE3.

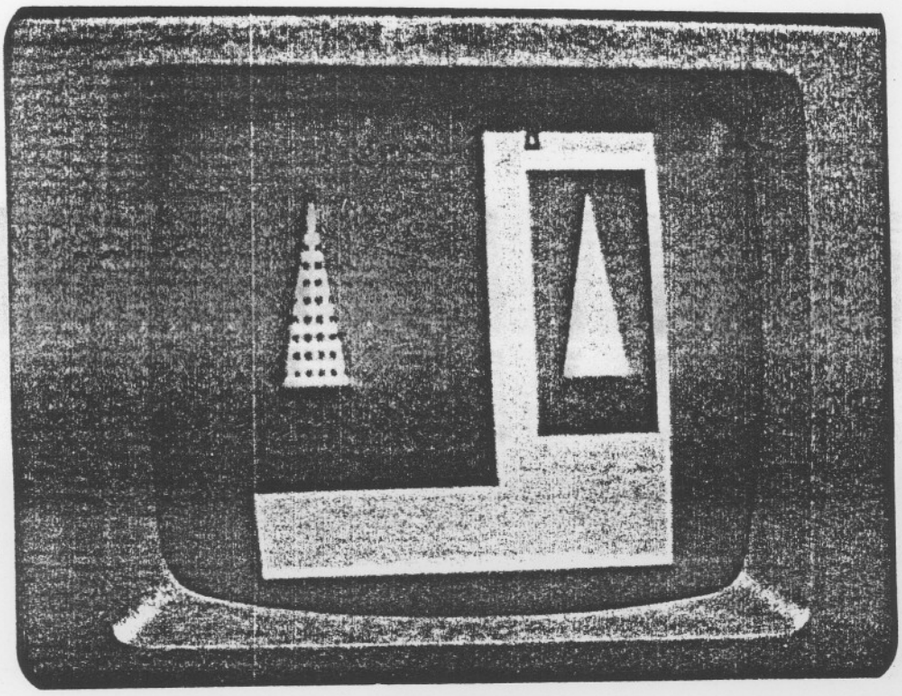


Fig.VI-1)- Vista do resultado do exemplo-1 na monitor gráfico.

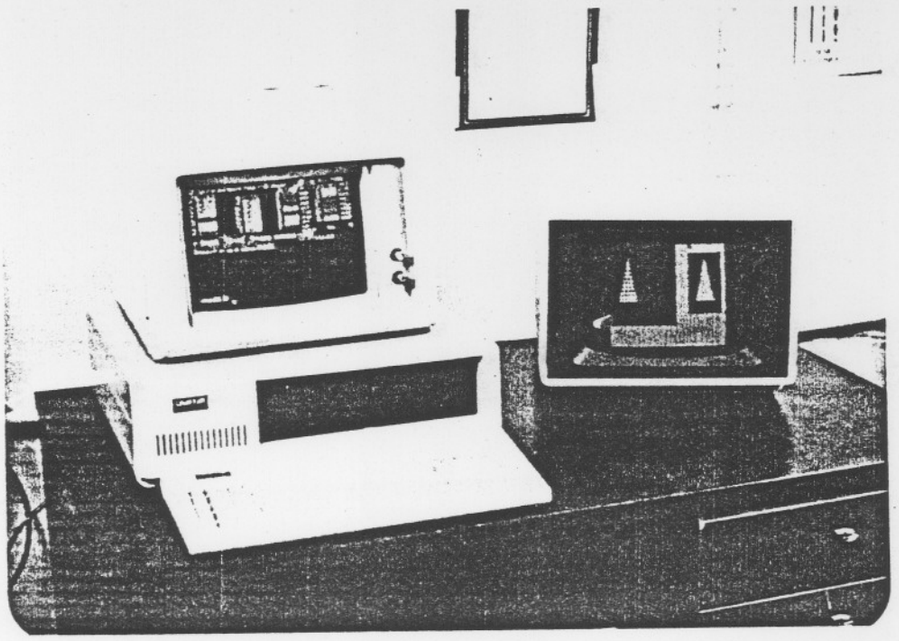
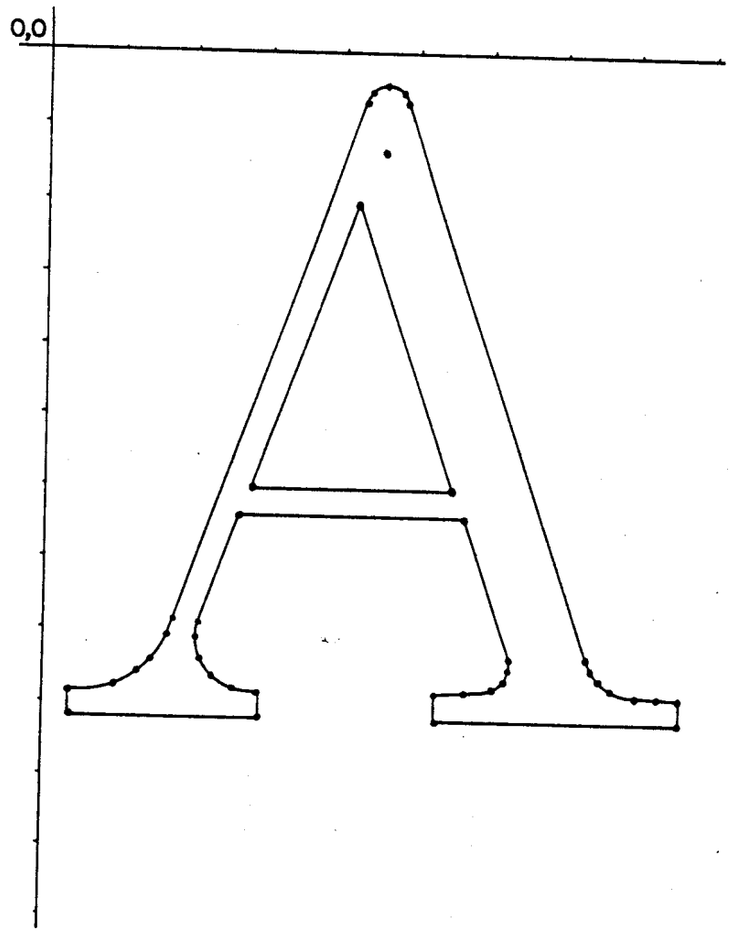


Fig. VI-2)- Vista da implementação física adotada nos testes.

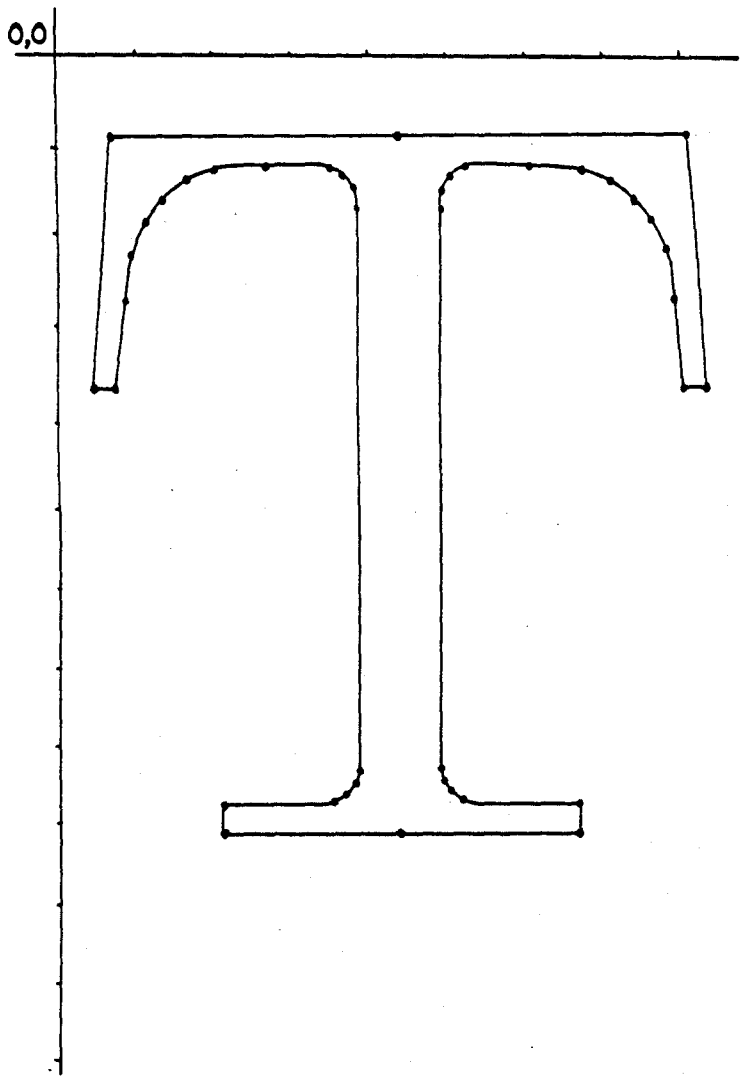
b)- Exemplo 2:

Neste exemplo, o usuário deseja definir as trajetórias para a confecção de caracteres, desenhados no estilo ROMAN (Knuth,86).
Serão utilizadas sómente retas.
O usuário vai definir a letra A (maiuscula), conforme o esquema abaixo:



```
GRAPHICS  FALSE  TRANSPARENCY  TRUE  BOUNDARY

(Entra no vocabulário gráfico e define alguns parâmetros)

: LETRAA 92 87 MOVETO 89 87 LINETO 89 81 LINETO 88 78 LINETO

         87 76 LINETO 85 75 LINETO 84 74 LINETO  7 48 LINETO

          5 47 LINETO  4 45 LINETO  5 43 LINETO  7 42 LINETO

         80 18 LINETO 82 17 LINETO 85 15 LINETO 86 14 LINETO

         87 12 LINETO 88  9 LINETO 89  4 LINETO 92  4 LINETO

         92 30 LINETO 89 30 LINETO 88 27 LINETO 87 24 LINETO

         85 22 LINETO 83 21 LINETO 81 21 LINETO 64 27 LINETO

         64 57 LINETO 84 64 LINETO 85 64 LINETO 87 63 LINETO

         88 61 LINETO 89 58 LINETO 89 54 LINETO 92 54 LINETO

         92 87 LINETO 60 28 MOVETO 60 56 LINETO 20 42 LINETO

         60 28 LINETO 13 45 MOVETO

         CURCPT

         0 91 MOVETO ;  (Fim da trajetória)
```

A outra letra desejada é a T (maiúscula).

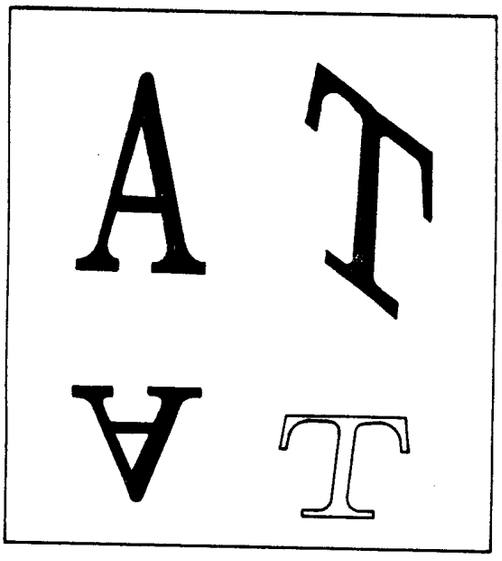



```
: LETRAT 92 67 MOVETO 88 67 LINETO 88 54 LINETO 87 52 LINETO
         86 50 LINETO 85 49 LINETO 83 49 LINETO 17 49 LINETO
         15 49 LINETO 13 51 LINETO 13 53 LINETO 13 61 LINETO
         14 68 LINETO 15 72 LINETO 16 74 LINETO 18 76 LINETO
         22 78 LINETO 28 79 LINETO 37 81 LINETO 37 84 LINETO
          9 81 LINETO 10 44 LINETO  9  7 LINETO 37  4 LINETO
         37  7 LINETO 28  8 LINETO 22 10 LINETO 18 11 LINETO
         16 13 LINETO 14 16 LINETO 13 20 LINETO 13 27 LINETO
         13 55 LINETO 14 37 LINETO 15 38 LINETO 17 39 LINETO
         83 39 LINETO 85 38 LINETO 86 37 LINETO 87 36 LINETO
         88 34 LINETO 88 21 LINETO 92 21 LINETO 91 44 LINETO
         92 67 LINETO 39 44 MOVETO
CURCPT
0 88 MOVETO ; (Fim da trajetória)
```

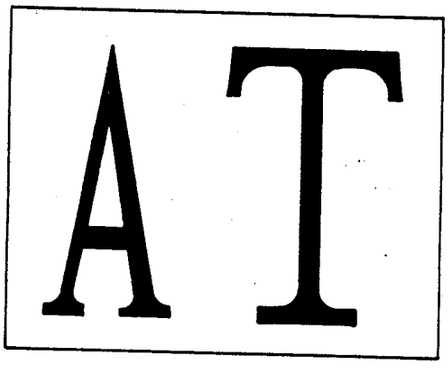
Definido os contornos, serão colocados nos Stencils.

```
4096 200 600 0 DOSTENCIL TELA            {Define TELA}
4096 150 300 2800 DOSTENCIL  MASCARA     {Define MASCARA}
BLACK TELA TELA COMPOSE                  {Inicializa TELA}
MASCARA MASCARA COMPOSE                  {Inicializa MASCARA}
REPLACE FILLMODE


MASCARA CURSTENCIL !                     {Define o Stencil
                                          corrente}
10 10 MOVETO CURCPT ORIGIN D!            {Posiciona}
TRUE LETRAA FILL                         {Desenha a letra A}
CURCPT  ORIGIN  D!  TRUE LETRAT FILL     {A partir da letra A,
                                          desenha a letra T}


{Define nova posição e escala agora no Stencil TELA}
TELA CURSTENCIL !
1 1 SCALEY D!
3 1 SCALEX D!
10 10 AMOVETO CURCPT ORIGIN D!
TRUE LETRAA FILL


{Rotaciona, escrevendo a letra T}
CURCPT ORIGIN D! -10 ROTATION TRUE LETRAT FILL


{Reposiciona os eixos, porém inverte o eixo Y}
0 ROTATION -2 3 SCALEY D!
190 10 AMOVETO CURCPT ORIGIN D!
TRUE LETRAA FILL


{Desinverte o eixo Y  e aplica distorção no eixo X}
100 300 AMOVETO CURCPT ORIGIN D!
2 3 SCALEY D! -1 3 SHEARX D!
{Escreve somente o contorno, usando BOUNDARY }
LETRAT
```

O resultado, em TELA, será:



O resultado em MASCARA:



Na foto abaixo, é mostrado o resultado na tela gráfica:

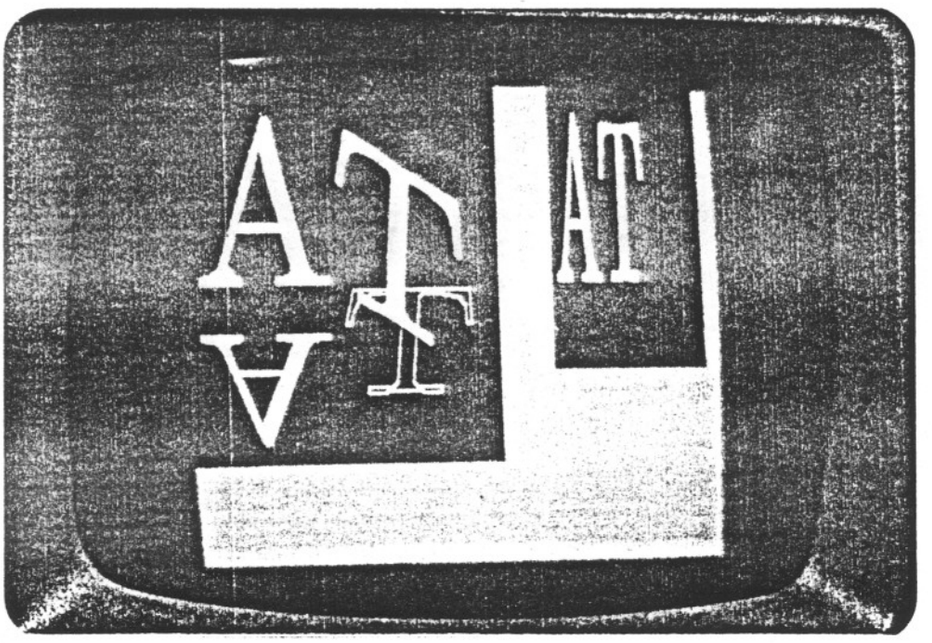


Fig.VI-3)- Exemplo 2, vista na tela gráfica.

c)- Exemplo 3:

O usuário deseja criar uma função gráfica, chamada CURVETO. Tal função será chamada da seguinte forma:

y1 x1 y2 x2 y3 x3 n CURVETO

e se utiliza dos pontos de controle y3x3, y2x2, y1x1, que a partir do ponto corrente (y0x0), traça uma curva aproximada a esses quatro pontos, em n segmentos de reta.

Matematicamente, tal aproximação se chama "Bezier Spline" de ordem 3 (Newman,81), (Enns,86)

Resumidamente, o comando CURVETO traça uma curva segundo as equações paramétricas:

$$X(t) = ax.(t)^3 + bx.(t)^2 + cx.t + x0$$

$$Y(t) = ay.(t)^3 + by.(t)^2 + cy.t + y0$$

com t variando entre 0 e 1.

Os coeficientes são calculados a partir dos pontos de controle fornecidos, a saber:

cx = (x1 - x0).3
cy = (y1 - y0).3
bx = 3.(x2 - x1) - cx
by = 3.(y2 - y1) - cy
ax = x3 - x0 - cx - bx
ay = y3 - y0 - cy - by

A curva desenhada por CURVETO é aproximada por n segmentos de reta.

Dessa forma, t assume os valores:

t = 1/n , 2/n, 3/n, ..., n/n

onde uma linha reta une cada ponto, correspondentemente.

Antes da criação do comando, por conveniência, o usuário irá definir todas as variáveis que serão utilizadas.

```
O VARIABLE AX
O VARIABLE BX
O VARIABLE CX
O VARIABLE AY
O VARIABLE BY
O VARIABLE CY
O VARIABLE X1
O VARIABLE Y1
O VARIABLE X2
O VARIABLE Y2
O VARIABLE X3
O VARIABLE Y3
O VARIABLE X0
```

```
0 VARIABLE Y0
0 VARIABLE N
0 VARIABLE N2
0 VARIABLE N3
0 VARIABLE T2
0 VARIABLE T3
```

A partir dessas variáveis auxiliares, a definição do comando fica:

```
: CURVETO
{Pega os dados e armazena}
N !  X3 !  Y3 !  X2 !  Y2 !  X1 !  Y1 !
VCPT D@ X0 !  Y0 !  {Veja função VCPT no apendice}

{Calcula os coeficientes}
X1 @ X0 @ - 24576 * CX !
Y1 @ Y0 @ - 24576 * CY !
X2 @ X1 @ - 24576 * CX @ - BX !
Y2 @ Y1 @ - 24576 * CY @ - BY !
X3 @ Y0 @ - CX @ - BX @ - AX !
Y3 @ Y0 @ - CY @ - BY @ - AY !
N @ DUP DUP * DUP N2 !  * N3 !

{Loop principal}
N @ 1 DO
      I DUP DUP * DUP T2 ! * T3 !

      {Calcula yt}
      Y0 @ CY @ I @ N @ * / +
      BY @ T2 @ N2 @ * / +
      AY @ T3 @ N3 @ * / + 8192 /        {Coordenada Y}


      {Calcula xt}
      X0 @ CX @ I @ N @ * / +
      BX @ T2 @ N2 @ * / +
      AX @ T3 @ N3 @ * / + 8192 /        {Coordenada X}

      {Traça o segmento da  linha}
      LINETO
LOOP
; {Fim da função CURVETO}
```

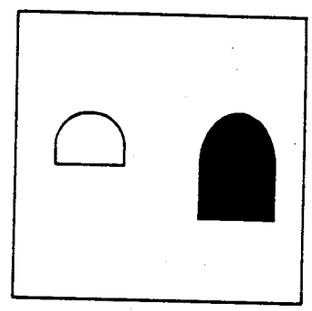
A multiplicação por 8192 visa a adicionar, aproximadamente, 5 casas decimais para aumentar a precisão das operações internas.

A multiplicação por 8192, é o mesmo que multiplicar por 2 elevado a 13, ou seja, deslocar o número binário 13 casas à esquerda. Dessa forma, a operação é muito rápida.

Exemplo de utilização:

```
{ cria uma trajetória com a forma desejada }
: JANELA 50   0  MOVETO
         30   0  LINETO
          0 0 0 15 0 25 8 CURVETO
          0 35 0 50 30 50 8 CURVETO
         50  50  LINETO
          0  50  LINETO
         25   25   MOVETO  CURCPT          {Deixa a semente para
                                            FILL}
          0   0  MOVETO
;

{  prepara  o  contexto gráfico,  usando  os  Stencils  definidos
anteriormente }
TELA CURSTENCIL !     TRUE BOUNDARY !     STROKE CURBRUSH !
DRAWMODE

{ posiciona }
30  30  AMOVETO  CURCPT  ORIGIN  D!

{ finalmente desenha }
JANELA

{ modifica o contexto gráfico }
FILLMODE
3  1  SCALEX D ! 2 1 SCALE Y D!
30   120   AMOVETO CURCPT ORIGIN D!      {Posiciona   e   muda   as
                                           escalas}
TRUE JANELA FILL
```

Em TELA, obter-se-á:



Nas fotos, abaixo, é mostrado o resultado na tela gráfica:

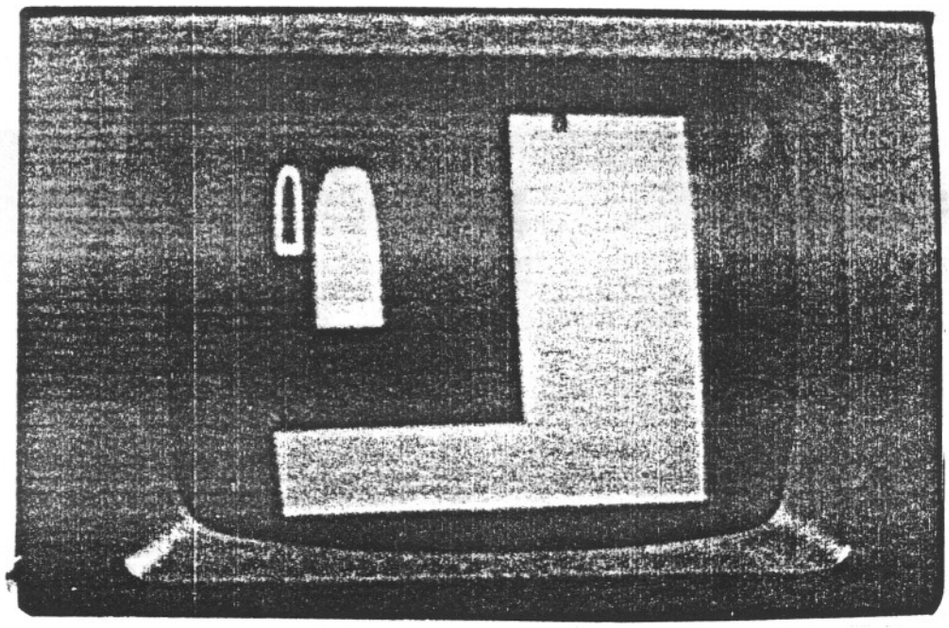


Fig.VI-4)- Exemplo 3

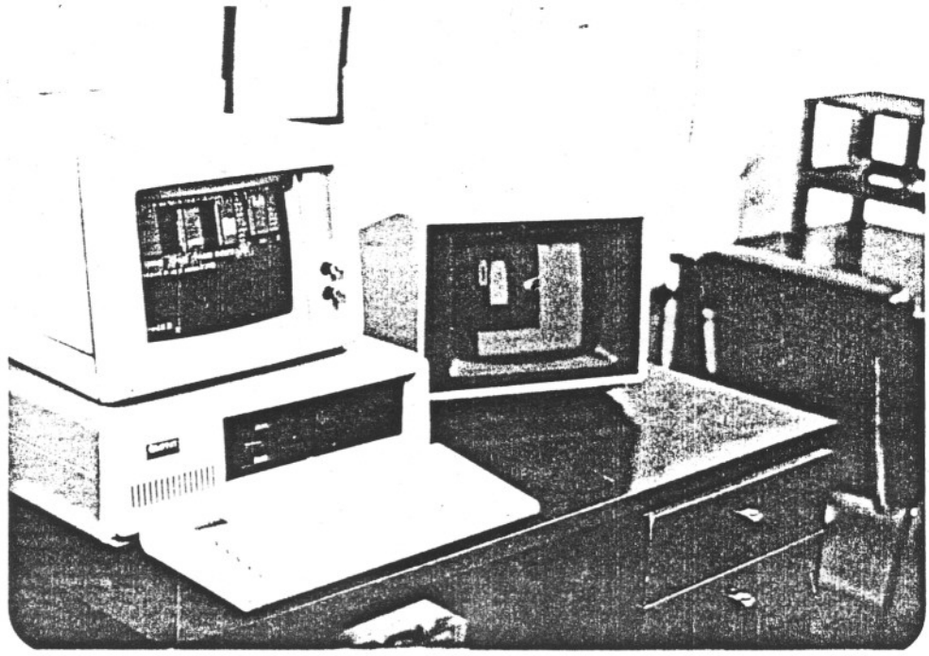


Fig.VI-5)- Exemplo 3

d)- Exemplo 4:

Novamente o usuário deseja criar uma função gráfica, desta vez denominada ARCTO.
Tal função é chamada da seguinte forma:

y x rdy rdx n ARCTO

Sua função: traçar um arco de circunferência a partir do ponto y x , cujo centro está deslocado rdy rdx do ponto inicial, por n graus. O ponto corrente é ligado ao ponto inicial por uma reta. Após a execução, o último ponto da curva é o ponto corrente.
Seu funcionamento baseia-se na seguinte transformação, conhecida de geometria analítica :

x' = x.cos(alfa) + y.sen(alfa)

y' = y.cos(alfa) - x.cos(alfa)

onde se rotaciona o ponto x,y de "alfa" graus com relação a origem.
Como a linguagem não possui números reais, deverão ser usados artifícios e aproximações.

O número PI=3.1415.... pode ser aproximado pela razão:

PI ~ 355/113

O valor PI/180 será, então:

PI/ 180 ~ 355/20340

Fatorando para um número múltiplo de 2 próximo:

PI/180 ~ 143/8192

o valor PI/180 equivale a 1 grau, então pode-se aproximar:

sen(1grau) ~ PI/180 ~ 143/8192

cos(1grau) ~ 1

A expressão para rotação do ponto y,x, de 1 grau com relação a origem, no sentido horário, é:

x' ~ x + y.143/8192

y' ~ y - x.143/8192

No sentido anti-horário, usando a equivalência:

sen(-1grau) = -sen(1grau)

Portanto:

$$y' = y - x.143/8192$$

$$x' = y.143/8192 + x$$

(Observa-se a troca y-x)

A multiplicação por 143/8192 pode ser simplificada, através das seguintes equivalências:

$$143 = 128 + 16 - 1 = 2^{7} + 2^{4} - 1$$

$$8192 = 2^{13}$$

Portanto, a transformação efetuada em 1 grau pode ser obtida por aritmética inteira, usando somente somas, subtrações, multiplicações e divisões por 2.

Desejando-se operar por mais que 1 grau, ou seja desejando-se n graus, basta repetir o processo n vezes.

Para girar no sentido anti-horário, basta detectar o sinal de n e intercambiar os valores y-x.

Tal algoritmo já está implementado na linguagem, através das funções internas da rotina ROTATION, chamadas:

RROTATION
ROTAVAR

y x RROTATION y' x' Pega o ponto y,x da pilha e rotaciona com relação ao ponto 0,0, pelo número de graus contido na variável ROTAVAR.

ROTAVAR Variável que contém o número de graus que será usado nas transformações subsequentes .

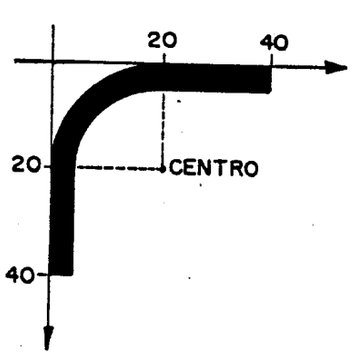
Para criar a função ARCTO, além de usarmos as duas rotinas internas anteriores, definiremos também as seguintes variáveis auxiliares:

```
O VARIABLE OLDROTAVAR     ( Armazena, temporariamente, ROTATION )
O VARIABLE BEGINARCX      ( Armazena ponto x inicial do ARCO )
O VARIABLE CENTERX        ( Armazena centro x da circunferencia )
O VARIABLE BEGINARCY      ( Armazena ponto y inicial do ARCO )
O VARIABLE CENTERY        ( Armazena centro y da circunferencia )
O VARIABLE RDX            ( Armazena posição relativa do centro )
O VARIABLE RDY            ( Armazena posição relativa do centro )
O VARIABLE N              ( Armazena número de graus )
```

```
: ARCTO
( armazena os dados )
N !  RDX !  RDY !  DUP  DUP BEGINARCX !  SWAP
DUP BEGINARCY ! SWAP LINETO

( encontra o centro )
BEGINARCX @ RDX @ + CENTERX !
BEGINARCY @ RDY @ CENTERY !

( prepara ciclo )
N @ 0 DO
     RDY @ NEGATE  RDX @ NEGATE
     OLDROTAVAR ! ! ROTAVAR !
     RROTATION
     OLDROTAVAR @ ROTAVAR !
     CENTERY @ CENTERX @ ADDXY
     LINETO
     LOOP
;
```

Exemplos de utilização:

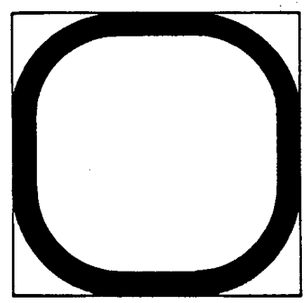
```
( define uma trajetória )
: CANTONEIRA  0  40  MOVETO
     0  20  20  20  90  ARCTO
            40  0  LINETO
             0  0  MOVETO
;
```



```
(Define uma função auxiliar)

: FIX CURCPT ORIGIN D ! ;

( desenha a figura )
TELA CURSTENCIL !
DRAWMODE
STROKE3 CURBRUSH ! COR
50 100  AMOVETO CURCPT ORIGIN D!
 2   1  SCALEX D !
 1   1  SCALEY D ! FIX
CANTONEIRA
50  260  AMOVETO CURCPT ORIGIN D!
-2    1  SCALEX D ! FIX
CANTONEIRA
130 260  AMOVETO CURCPT ORIGIN D!
 -1   1  SCALEY D ! FIX
CANTONEIRA
130 100  AMOVETO FIX
  2   1  SCALEX D!
CANTONEIRA
```

Observa-se que a rotação em multiplos de 90 graus foi feita somente através do comando SCALEX , SCALEY.

Em TELA obter-se-á:



Na literatura, existem uma série de outras propostas para a implementação de arcos de circunferência, elipses, etc, sempre usando aritmética inteira (VanAken,84), (Pitteway,67).

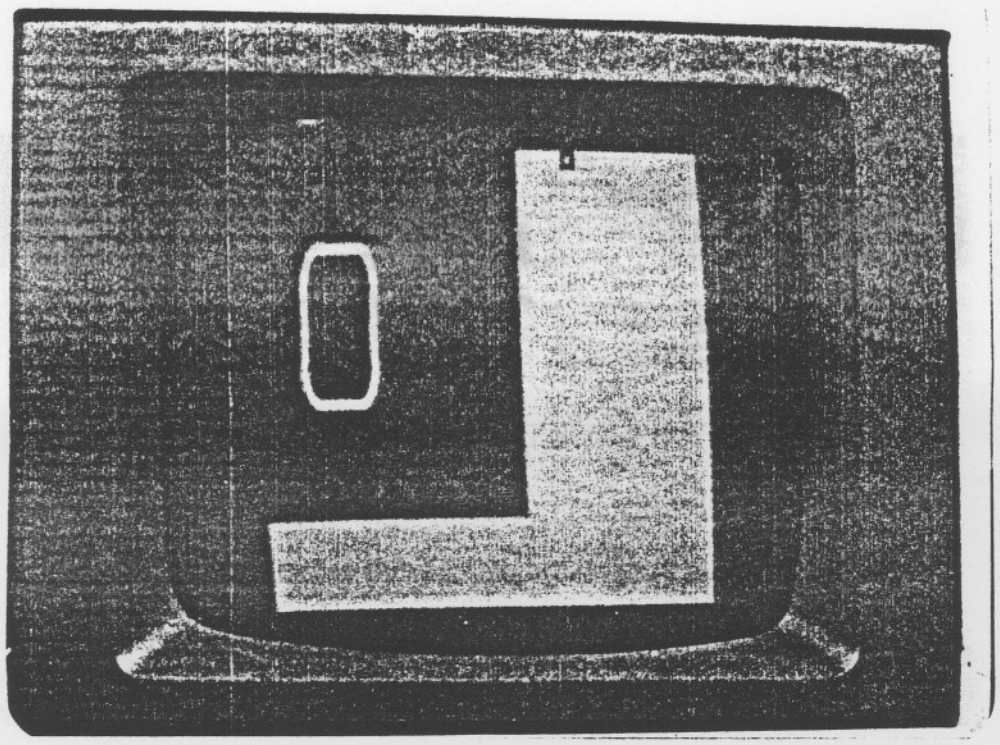


Fig.VI-6)- Exemplo 4

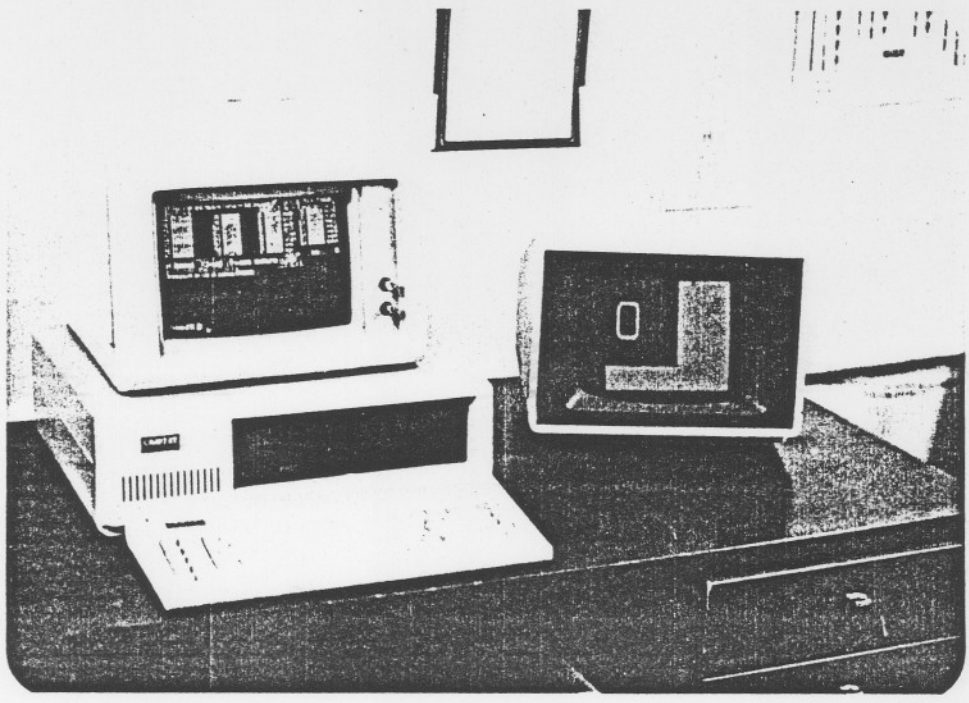


Fig.VI-7)- Exemplo 4

e)- Exemplo 5:

Neste exemplo, será explorada a característica de recursividade da linguagem, em uma aplicação gráfica. A função desejada é:

y x t r ARVORE

Trata-se de uma trajetória, que automaticamente muda seus parâmetros e se chama novamente, formando uma "lista" de trajetórias encadeadas.

A trajetória básica consta em, simplesmente, traçar uma reta do ponto y x fornecido até um ponto situado em coordenada relativa ao primeiro, a uma distância t no rumo r. Ao se chegar a esse ponto , diminui-se t em 2/3 e ordena-se a traçar novamente a trajetória duas vezes, a primeira a um rumo de r + 30 graus, a segunda a r - 30 graus. O processo continua, até que t seja menor que 5.

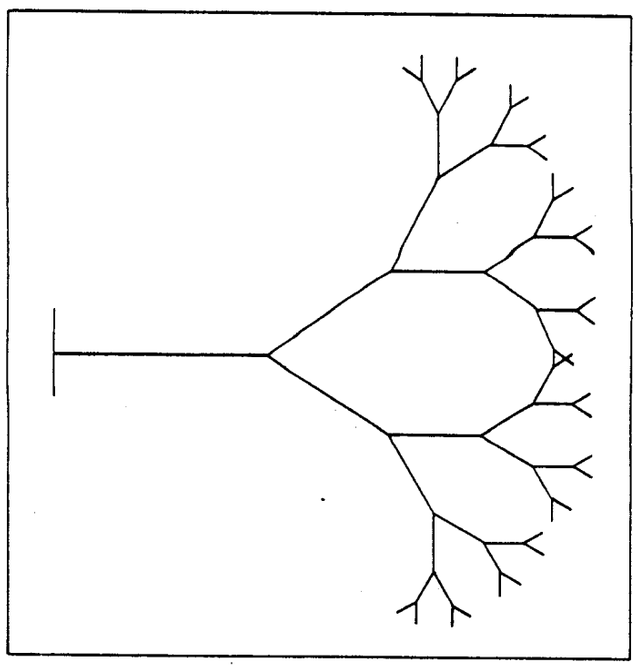
A definição da função é a seguinte:

```
{Variáveis auxiliares}
DOUBLEVAR NEWPOINT  {Armazena ponto Y X fornecido}
0 VARIABLE NEWPOINT   {Armazena novo rumo fornecido}
0 VARIABLE NEWSIZE    {Armazena novo tamanho fornecido}

: ARVORE
{Salva dados}
NEWROT ! NEWSIZE ! NEWPOINT D!

{Verifica se o tamanho é maior que 5}
NEWSIZE @ 5 > IF
          {Se TRUE traça}
          NEWPOINT D@ AMOVETO
          NEWROT @ ROTATION !
          NEWSIZE @ 0 RLINETO
          {Calcula nova trajetória}
          NEWSIZE @ 2 3 */ NEWSIZE !
          {Prepara para nova chamada}
          CURCPT NEWSIZE @ NEWROT @ 30 +
          CURCPT NEWSIZE @ NEWROT @ 30 -
          {Chama a função novamente}
          ARVORE
          ARVORE
THEN
;
```

Um exemplo de utilização.

```
TELA CURSTENCIL !
FILLMODE
1  1  SCALEY D!
1  1  SCALEX D!

50  100  20  90  ARVORE
```

Em TELA, obter-se-á:



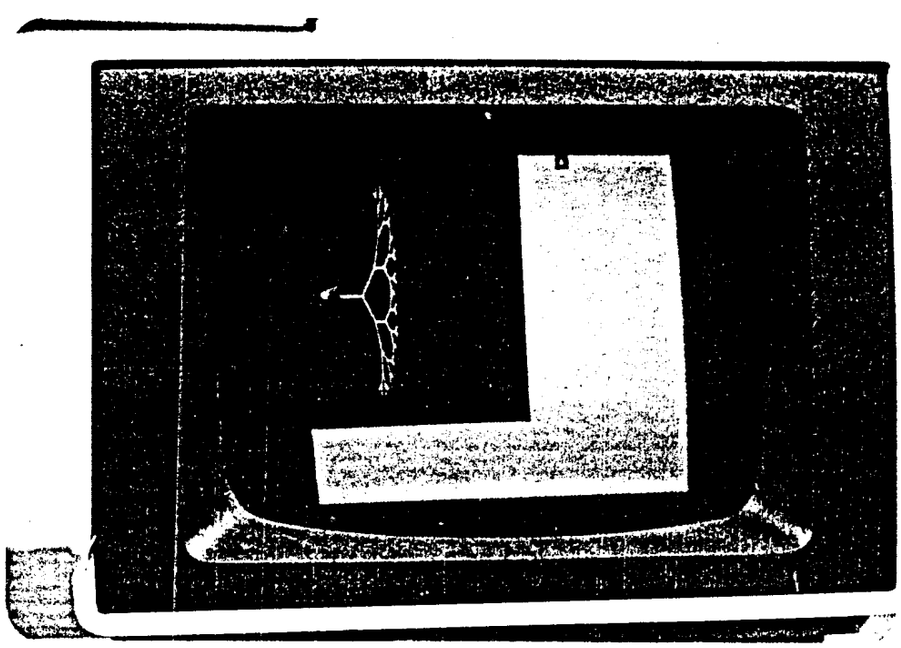


Fig.VI-8)- Exemplo 5

VI-13) Alguns comentários sobre a implementação

O conjunto de comandos aqui apresentados, compõem um núcleo poderoso e ao mesmo tempo simples. Tal núcleo é totalmente independente da escolha do mecanismo ,ou hardware impressor, onde está implantado . Para que os stencils com a imagem desejada sejam enviados para o canal impressor, é necessário, porém, que existam algumas informações e comandos que saibam como se dialogar com o mecanismo.

Nesta implementação , o comando responsavel a transmitir um stencil para o canal de impressão, dentro da máquina, é chamado de "SHOWPAGE".

Por exemplo, o usuário deseja visualizar o stencil TELA:

TELA SHOWPAGE

Este comando é escrito, usando as informações específicas do mecanismo utilizado e , portanto, específico de cada implementação e não será abordado aqui.

Os sinais típicos de interfaceamento foram discutidos no Capítulo II, na descrição do controlador de escrita.

No capítulo seguinte, será descrito a implementação física adotada para os testes e avaliação do modelo adotado.

VII)- IMPLEMENTAÇÃO FISICA ADOTADA

VII.1)- Introdução

Para se efetuar os testes na linguagem e comparar o funcionamento das estruturas propostas, seria interessante que existisse alguma impressora que possibilitasse a introdução do sistema.

Infelizmente tal ação, no momento, não é viável, pois as impressoras laser disponíveis não permitem essa troca.

Este fato obrigou o autor ao uso de simulações. A simulação adotada baseou-se na existência de uma placa expansora gráfica para o microcomputador PC. Tal placa possui o TMS34010, memória, interface serial própria e permite o funcionamento autonomo com relação ao PC. Esta placa chama-se SDB340 e é fabricada pela Texas Instruments.(Texas,87)

VII.2)- A Placa SDB340

A placa SDB340 e uma placa de expansão para o IBM PC, conectado diretamente ao Duto de Expansão deste. Esta placa possui o processador TMS34010, correndo a 40MHz, 512K Bytes de memória RAM, 256K Bytes de memória de vídeo e uma interface serial própria.

Esta placa possui também interface RGB e conexões para vídeos gráficos de alta resolução, liberando o vídeo do PC.

Junto com a placa acompanham programas depuradores, que permitem ao usuário testar e corrigir sua aplicação .

O Depurador é altamente interativo e permite a monitoração do que ocorre nesta placa através do PC, permitindo ao usuário acompanhar todo o processamento interno da aplicação, registradores, interrupções, etc.

Maiores detalhes sobre a placa encontram-se nas referências (Texas,86),(Texas,87).

VII.3)- Implementação Física

Para os testes foi utilizado um microcomputador IBM PC com 640K e interface serial.

A placa SDB340 foi configurada para acionar um monitor de vídeo ITAUTEC , monocromático, com a resolução de 600 x 400 pontos.

Na simulação feita, o sistema era carregado através do PC, através do programa depurador da Placa SDB e também inicializado.

A partir daí, o Miniscript passava a aceitar comandos pelo interface serial, simulando dessa forma uma "pseudo-impressora".

O "papel" sendo impresso é visto no "vídeo gráfico" , enquanto no monitor do PC se verifica o desenrolar do processamento através dos comandos do depurador.

Como o Miniscript somente aceita comandos pela interface serial da placa SDB, era necessario para o funcionamento do sistema um outro microcomputador. Este enviaria os comandos pela interface como se estivesse trabalhando com uma impressora.

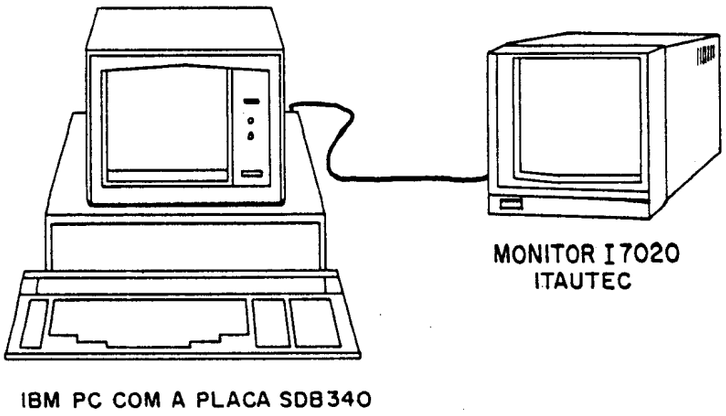


Fig.VII-1)- Implementação física adotada para os testes.

Porém, foi possível, também, a utilização da própria interface serial do PC, na qual estava ligado a placa SDB . Para transmitir os comandos ao Miniscript , o usuário acumulava os comandos num arquivo, sendo enviado à interface através do comando "PRINT" do sistema DOS 3.0. Tal estratégia, apesar de simples, tem seu custo na velocidade de processamento, visto que o comando "PRINT" compartilha execução com o programa depurador do SDB340.

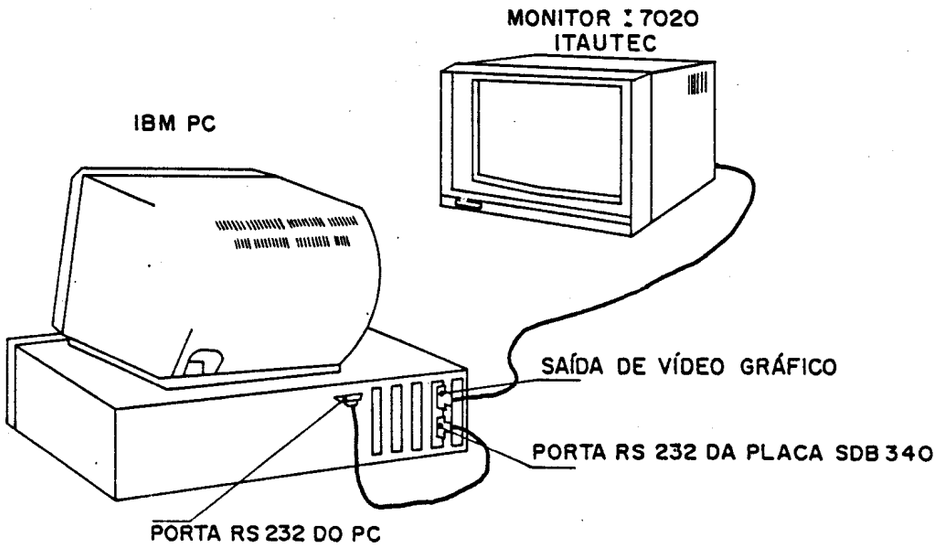


Fig.VII-2)- Conexão interface serial PC - Interface serial da Placa SDB340, usando um único computador.

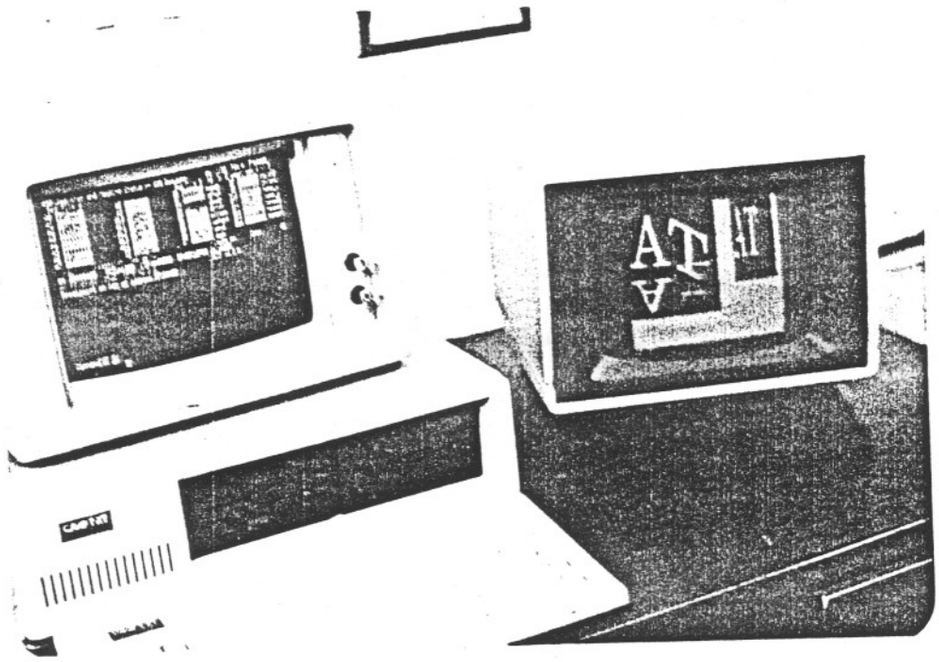


Fig.VII-3)- Vista da implementação física adotada nos testes.

VII.4)- Adaptações Necessárias

Como discutido no Capítulo V, existem 4(quatro) rotinas que são as interfaces entre o software e o hardware utilizado.

As rotinas INPUT e OUTPUT foram escritas para acessar a interface serial (USART) da placa SDB340 e também efetuar o "handshake".

A rotina ABORT é responsável pela " partida a frio ", inicializando algumas variáveis do sistema e também algumas configurações internas do SDB340.

A rotina SHOWPAGE foi escrita visando acessar a saída de vídeo do SDB340.

Tais rotinas estão apresentadas no apêndice, na listagem geral do sistema implementado.

VIII)- AVALIAÇOES DE DESEMPENHO

VIII.1)- Introdução

Para se verificar que a implementação proposta possue as qualidades alegadas , é necessário que se façam programas testes que explorem as virtudes e defeitos do sistema em questão.

No caso do Miniscript, o ideal seria a de operar duas impressoras. Em uma impressora instala-se o Miniscript e na outra, igual, uma linguagem conhecida e tida como padrão, por exemplo o Postscript.

Infelizmente, como não se tinha disponível uma impressora possível de se instalar o Miniscript ,tal comparação não pode ser feita.

Pensou-se, então, em processar o mesmo tipo de exemplos e medir-se o tempo, porém tal procedimento não seria exato, pois seria muito dificil comparar o tempo que uma imagem ficava pronta na tela gráfica em comparação com um tempo gasto por uma da impressora.

Foi então decidido fazer algumas comparações baseado em dados disponíveis nas referências. Obviamente não se deve tomar como absolutas tais comparações, pois diversos aspectos relevantes estão agindo ao mesmo tempo. As comparações devem ser tomadas mais no sentido de ressaltar as diferenças estruturais básicas e assim inferir prováveis virtudes e defeitos . Ressalta-se também que a implementação aqui feita , o Miniscript, constitui-se num núcleo básico, e como toda linguagem "TIL" , considerada inacabada. Esta consideração advém do fato de que o usuário deve criar , ou "afinar" , a implementação à sua necessidade, criando as funções de conveniência que diminuam a "aspereza" do sistema original.

Será tomada como base de comparação a linguagem Postscript, e serão analizados aspectos como tamanho do código, recursos , velocidade etc.

VIII.2)- Tamanho do código

O Postscript tem uma série de implementações, dependendo do equipamento utilizado. Nas referências foi obtido um tamanho médio para uma configuração comum, encontrada nas impressoras a laser de pequeno porte.

A implementação mais comum do Postscript ocupa 512 KBytes para o código e neste espaço estão incluídos 13 tipos diferentes de fontes de caracteres (definidos por "outline" , ou contorno) (Adobe,85).

A memória RAM encontrada varia de 1MByte até 3MBytes, sendo que o Postscript usa cerca de 200KBytes, como área de rascunho, para expandir os caracteres de "outline" para "bit-map", etc. O restante é destinado à montagem das páginas.

O Miniscriipt, aqui implementado, tem a dimensão de 11KBytes, porém sem incluir os fontes de caracteres. No Miniscript, um fonte completo (126 caracteres), definido por contorno (trajetória), ocupa cerca de 2KBytes cada, o que daria um valor de 26 KBytes para se repetir os 13 fontes do Postscript. O total seria, portanto, cerca de 36 KBytes.

A memória RAM também deve se situar, no caso do Miniscript, entre 1 e 3 MBytes, porque este número é o mínimo necessário para representar uma pág ima comum na resolução de 300 dpi. A área de rascunho que o Miniscript necessita também é da ordem de 200 KBytes, pela mesma razão anterior.

Pode-se verificar, portanto, que o código do Miniscript é cerca de 8 vezes menor. A principal razão desta compactação é o exaustivo uso da filosofia e estruturas das linguagens TIL.

O Postscript, apesar de lembrar muito a linguagem FORTH não é uma linguagem TIL, como seus próprios autores enfatizam. O Postscript foi codificado numa linguagem de alto nível e compilada (Adobe,85).

O Miniscript foi escrito para o processador TMS34010, e o Postscript para o Motorola 68000/68020.

O fato de o Miniscript ter sido escrito um processador gráfico pode ter reduzido a necessidade do código, pois certas funções existem diretamente no processador.

Torna-se, portanto, necessário uma comparação dos recursos.

VIII.3)- Recursos Existentes no Modulo Basico.

O fato de se comparar somente o tamanho do código não é válido sem que nos detenhamos nos recursos ali embutidos.

Na tabela abaixo, será usado a seguinte convenção:

S : Possui

N : Não possui

P : Não possui mas pode ser implementado pelo usuário

- : Não se aplica

CARACTERISTICAS	POSTSCRIPT	MINISCRIPT
Fontes ou conjuntos tipográficos		
Fontes definidas por "OUTLINE" (Contorno)	S	S
Fontes definidas por "BITMAP" (Conj. de Pontos)	S	S
Rotação de caracteres	S	S
Translação de caracteres	S	S
"Scaling" de caracteres	S	S
Inclinação de caracteres	P	S
Espacamento variável	S	S
Possibilidade do usuário criar novas Fontes	S	S
Fontes embutidos	S	P
Funções gráficas		
Capacidade do usuário criar trajetorias	S	S
Trajetorias em sistema de coordenadas absoluto	S	S
Trajetorias em sistema de coordenadas virtual	S	S
Linhas de espessura variável	S	S
Pinceis	S	S
"Filling", ou pintura de regiões delimitadas	S	S
Capacidade de criar novas estruturas gráficas	N	S
Capacidade de composição de imagens	S	S
Capacidade de criar desenhos, logotipos, etc	S	S
"Raster Operations", ou Operações "Raster"	N	S
Capacidade de trabalho com vários Stencils páginas ou máscaras ao mesmo tempo	N	S
Capacidade de processar imagens prontas enviadas até a impressora	S	P
Funções para traçagem		
Funções de conveniência para o acabamento de linhas (canto vivo , canto redondo , pontudo etc)	S	P
Arcos , Círculos	S	S
Interpolações	S	S
Concordância automática entre retas e círculos	S	P
Escala de cinza	S	P
Capacidade de cores	S	P

Estruturas de controle

IF, THEN, ELSE (ou equivalente)	S	S
BEGIN, WHILE, REPEAT (ou equivalente)	S	S
DO LOOP (ou equivalente)	S	S
Capacidade de criar novas estruturas	N	S
Detecção de estruturas mal posicionadas	S	P

Funções, Sub-rotinas

Capacidade de se criar novas funções ou subrotinas	S	S
Capacidade de se criar funções recursivas	S	S
Capacidade de se reescrever funções do sistema	N	S
Capacidade de se organizar funções em grupos (vocabulários)	S	S

Estruturas de dados disponíveis

Constantes inteiras	S	S
Constantes reais	S	P
Variáveis inteiras	S	S
Variáveis reais	S	P
Matrizes	S	S
Vocabulários (ou equivalentes)	S	S
Capacidade de se criar novas estruturas de dados	N	S
Verificação dos limites estruturais nas variáveis	S	P
Verificação dos limites em matrizes	S	P
Pilhas	S	S
Verificação dos limites nas pilhas	S	S

Funções aritméticas

Funções inteiras	S	S
Funções racionais	P	S
Funções reais	S	P
Funções logaritmas, exponeciais, senoidais	S	P
Operações lógicas	S	S
Operadores matriciais	S	P
Detecção de erros em funções aritméticas	S	P

Outras características

Modo compilativo	N	S
Capacidade de interações com o usuário	S	S

Com base no texto acima, pode-se concluir que o Miniscript leva vantagens na capacidade de expansão, podendo o usuário criar as estruturas de dados, gráficos e de controle que necessitar. O Miniscript possui uma grande capacidade de processamento gráfico, graças aos "Raster Operations" que o processador TMS34010 efetua. Outra vantagem é a de se poder trabalhar com vários "Stencils".

O Postscript a leva vantagem de que a linguagem já incorpora uma série de funções de conveniência, tais como : concordância de arco e reta, modificadores de retas, etc. Outra diferença é a de possuir aritmética real, o que pode facilitar a implementação de algumas funções de processamento numérico que se utilizem de funções periódicas como Seno, Cosseno, etc.

O Postscript também exerce um monitoramento mais profundo na checagem das operações desejadas pelo usuário, ou seja, um maior controle de erros.

Como estamos procurando comparar o núcleo de linguagens e não as funções de conveniência , concluímos que as duas linguagens possuem recursos e possibilidades muito semelhantes, com ligeira vantagem ao Miniscript que permite a criação de estruturas dedicadas a necessidade do usuário. O controle de erros é maior no Postscript tornando a linguagem menos suceptível a erros de programação . Porém , a forma modular do Miniscript permite que as definições introduzidas pelos usuários sejam facilmente testadas , além de também permitir ao usuário criar por ele mesmo os testes automáticos necessários, tornando sua aplicação específica , segura contra erros.

VIII-4) Velocidade

Como já discutido, é difícil uma comparação direta de velocidade entre o Miniscript e o Postscript, pois, além de possuírem estruturas muito diferentes, não foi possível a utilização de um mesmo hardware impressor, onde as linguagens pudessem ser instaladas e comprovadas. No sistema físico adotado para os testes, descrito no capítulo anterior, o processamento era compartilhado entre o depurador e o programa "PRINT" do DOS, o que também não permitia a medida do tempo de processamento dos exemplos.

Assim, para se ter uma idéia da velocidade do Miniscript em algumas operações , mediu-se, através do programa depurador, o número de ciclos de máquina gastos para executar certas tarefas:

Assim , tem-se em média :

-tempo para procurar uma palavra no modo interpretativo pelos vocabulários corrente , de contexto e até o vocabulário raiz : 6k ciclos;

-tempo para pegar uma palavra , procurá-la pelo dicionário , verificar que não foi definida e transformar num número : 8 Kciclos;

-tempo para compor dois stencils de tamanho 200x200, incluindo tempo de busca no dicionário : 4kciclos

Considerando que o TMS34010 pode andar a 40Mhertz, pode-se concluir que tais operações demoram muito pouco.

No Miniscript, após uma definição ser compilada, os tempos de execução são muito baixos. O Postscript não possui o modo compilativo, sendo a execução sempre no modo interpretativo, tornando lenta a execução de aplicativos do usuário.

VIII-5) Conclusões sobre as avaliações

Apesar de o Miniscript não ser em sua forma básica tão extenso quanto o Postscript , a sua capacidade de expansão permite que se crie novas estruturas e comandos alcançando os recursos disponíveis na outra. O Miniscript possui toda a base necessária para , sobre ela, se desenvolverem aplicativos sofisticados . Por usar de uma estrutura linguística radicalmente diferente , seu tamanho de código é no mínimo 8 vezes menor e por se utilizar de um processador gráfico , seu tempo de execução de comandos dessa categoria é muito reduzido. Tal fato, aliado à estrutura de linguagem em si e especialmente à existência do modo compilativo, faz com que o Miniscript seja potencialmente rápido na execução dos aplicativos do usuário.

Como o Miniscript é uma linguagem visando controlar dispositivos impressores , normalmente esta residente na impressora. Assim muitas vezes , quem envia os dados a ela são os programas aplicativos do usuário , tais como editores de texto , "Desktop Publishing Systems" , etc. Sendo assim , tais programas podem configurar e criar estruturas dedicadas aos trabalhos a que se propõe , tornando o sistema mais objetivo em sua tarefa final.

IX)- CONCLUSOES FINAIS

Como exposto nos capítulos iniciais, as características que a linguagem proposta deveria possuir, podem ser resumidas como:

- ser de fácil assimilação;
- ser expansível;
- ser rápida;
- ser independente do hardware impressor.

O modelo gráfico adotado, através de entidades que simulam os "stencils", "pincéis", "formas" e "trajetórias", fazem com que o usuário raciocine "fisicamente", encontrando a melhor forma de resolver o seu problema específico. Este fato também facilita a assimilação da estrutura da linguagem e seus comandos. Todos os comandos são mneumônicos, facilmente inteligíveis.

A estrutura de linguagem é totalmente expansível, podendo inclusive o usuário criar o seu próprio "sistema operacional", novas estruturas de dados e de controle, tipos de variáveis, etc.

Ressalta-se, que como o Miniscript é uma linguagem visando residir na impressora, normalmente quem envia os dados a ela são os aplicativos, que são executados no computador mestre, tais como: os "Desktop Publishing System", "CAD", etc.Tais aplicativos podem criar e configurar, no Miniscript, uma grande série de estruturas e comandos dedicados ao seu objetivo,tornando o sistema como um todo mais eficiente.

Os elementos criados pelo usuário são chamados pelo nome que ele mesmo atribuiu, tornando assim a sua aplicação facilmente legível.

A estrutura modular da linguagem permite fácil depuração, bastando testar os módulos independentemente.

A existência dos modos interpretativo e compilativo faz com que o sistema não perca tempo com buscas desnecessárias de endereços de execução, visto que em uma definição criada, todos os endereços já foram "compilados".

Tal fato, aliado ao uso do processador gráfico TMS34010, permite que o sistema tenha grande velocidade de execução em procedimentos gráficos.

A estrutura de linguagem e o uso do processador gráfico permitiu, também, uma grande compactação, haja visto que as capacidades do Miniscript são próximas ao do Postscript, com um código cerca de 8 vezes menor.

Todas as estruturas descritas são independentes do mecanismo impressor, visto quo somente 4 rotinas são "pontes", efetuando a comunicação necessária.

Dado o exposto, pode-se concluir que a linguagem proposta possui características interessantes, que podem viabilizar seu uso numa gama de máquinas impressoras não impacto, principalmente nas impressoras laser de pequeno porte. Sua utilização também pode ser viável em terminais gráficos inteligentes, ou "mini workstations", abrindo uma nova possibilidade para os computadores pessoais.

X)- SUGESTOES DE AMPLIAÇAO , APERFEIÇOAMENTOS E TRABALHOS FUTUROS

Serão discutidos a seguir algumas sugestões do autor , para trabalhos futuros , visando o aperfeiçoamento do sistema e mesmo suprir certas deficiências observadas.

O Autor sugere a inserção no núcleo básico, de alguns comandos que não foram aqui implementados , porém são de fácil criação pelo usuário .

- Criação de um comando "FORGET" , que serviria para cancelar uma definição já feita. O sistema cancelaria todas as definições existentes no dicionário após a definição indesejada. Este procedimento seria feito , seguindo os elos da lista encadeada até a palavra em questão. Por isso, o cancelamento não respeitaria os vocabulários , agindo diretamente na posição física do dicionário. Seria necessário criar uma rotina para o reestabelecimento de elos apagados nos vocabulários , retrocedendo até endereços físicos anteriores a palavra apagada. Exitem várias propostas para essa rotina na literatura. (Ting,86),(Brodie,87);

- Criação de um comando "IMAGE" , que permitiria o recebimento pela interface de comunicação de imagens já prontas ou compactadas , visando a impressão de fotos , imagens recolhidas por equipamentos especiais , etc . Na literatura são sugeridas várias estratégias de transmissão e compactação (Newmam,81),(Rogers,85);

- Criação de um comando "ROTATE-BIT-MAP" , para permitir a rotação de caracteres definidos em stencil , e mesmo stencils. Sem dúvida uma rotina complicada devido a necessidade de implementar procedimentos de redução de erros , "anti-aliasing" etc (Newman,81), (Crow,78), (Casey,82);

- Criação de comandos "CURVETO" , usando outros algorítmos além do de "BEZIER" , pois neste a escolha dos pontos de controle são difíceis e críticas (Newman,81),(Rogers,85);

- Opcionalmente , o sistema poderia possuir o assembladar interno , como nas linguagens FORTH , para permitir o usuário criar rotinas otimizadas em velocidade. Na literatura existem vários exemplos deste tipo de implementação , aliados a editores, depuradores etc. (Loelinger,81)(Ting,86)(Brodie,87)(Derick,82);

-E interessante que o sistema possua alguns "fontes" já residentes no núcleo básico , para melhorar seu desempenho;

- Para permitir que o sistema seja armazenado em memórias ROM , as variáveis devem possuir um ponteiro que as façam buscar seus dados respectivos na memória RAM. Este ponteiro serviria para a geração das variáveis no momento de sua criação. Na literatura existem várias propostas (Loelinger, 81), (Ting, 86), (Derick, 82);

- Há necessidade de criação de comandos que permitam o uso de pinceis "girantes", para aconpanharem as nuances de curvas que , por exemplo, existem em caracteres japoneses;

- A rotina SHOWPAGE deve prever a existência de sinais provenientes da impressora , tais como sinais de sincronismo , tipo de bandeja , tamanho de papel , erros etc.

- Seria também interessante a abreviação de certos comandos, evitando a transmissão de muitos caracteres no modo interpretativo. Por exemplo:

: M MOVETO ;

No exemplo, podemos inserir:

O O M

ao invés de:

O O MOVETO

XI)- REFERENCIAS BIBLIOGRAFICAS

As referências, abaixo relacionadas, são citadas em uma ou mais localidades dentro do texto . Para mencioná-las, foi utilizada a seguinte convenção:

("sobremone do autor" ,"ano da publicação")

Por exemplo ,uma citação a referência seguinte

ABE , F. et alli
High Speed Laser Printer With High Resoluction/Proceedings
Laser 79 Opto-Electron Conference Munich , July 2-6, 1.979
IPC Science and Technology Press Surrey Eng. -1979

poderia ser:

(Abe,79)

Quando houver mais de um autor , se utilizará somente do primeiro, em ordem alfabética .
Quando houver mais de um trabalho de mesmo autor , o discriminador será o ano. Se houver mais de um no mesmo ano , será utilizado o índice de ordem (1, 2, 3, etc) em seguida o ano.
Se a referência for de um trabalho anônimo , "Anon" será o nome, seguido do ano e número de ordem.

A seguir, é apresentada a relação em ordem alfabética:

(Abe,79)
ABE , F. et alli
High Speed Laser Printer With High Resolution/Proceedings
Laser 79 Opto-Electron Conderence Munich , July 2-6, 1.979
IPC Science and Technology Press Surrey Eng. -1979

(Adobo-1,85)
ADOBE SYSTEMS INC.
Postscript Language Reference Manual
Addison Wesley Publishing Company 1985

(Adobe-2,85)
ADOBE SYSTEMS INC.
Postscript Language Tutorial and Cookbook
Addison Wesley Publishing Company 1985

(Anon,86)
ANON,.
Laser Printers Approach Pre-eminent Position
Laser Focus - November 1.986 - pg. 36-39

(Archibald,82)
ARCHIBALD, R.
Laser Printer Fusing System
Hewllet Packard Journal - June 1.982 - pg. 24-26

(Baker,83)
BAKER, T. et alli
Graphics Capabilities on a Laser Printer
Hewllet Packard Journal - November 1.983 - pg. 17-22

(Barrera,84)
BARRERA, C. AND STRIETZEL, A. V.
Electrophotographic Printer Control as Embodied in the IBM 3800 Printing Subsystem Model 3 and 8
IBM J. Res. Development - Vol. 28 - n.3 - May 1.984 - pg. 263-275

(Barret,87)
BARRET, J. and REISTROFFER, K.
Designing a Raster Image Processor
Byte - May 1987 - pg.171-180

(Baumann,84)
BAUMANN, G.
Flash Fusing in Electrophotographic Machines
IBM J. Des. Development - Vol. 28 - n.3 - May 1984 - pg.292-299

(Benda,81)
BENDA, J.
A Model for Magnetic Brush Development in Xerographic Machines
IEEE Transactions on Industry Applications - Vol. IA-17 - n.6 - November December 1.981 - pg. 610-618

(Bergmam,76)
BERGMAM, V. et alli
Focus Path of a Laser Beam Deflected by a Prismatic Polygon Mirror: Its Calculation and Optimization
Applied Optics - Vol. 15 - n.12 - December 1.976 - pg. 3084-3088

(Bernadelli,84)
BERNADELLI, W. et alli
Tangential Field Production in Electrophotographic Developers
IBM Technical Disclosure Bulletin- Vol. 26- n.8 - January 1.984 - pg. 4407-4408

(Bhatt,86)
BHATT , S. AND PRESTIGIACOMMO T.
Page Description Languages
Computer Graphics World - September 1.986 - pg.79-84

(Borch,84)
BORCH, J.
Paper Material Considerations for System Printers
IBM J. Des. Development - Vol. 28 - n.3 - May 1984 - pg. 285-291

(Bresenham,65)
BRESENHAM, J.E.
Algorithm for Computer Control of a Digital Plotter
IBM Systems Journal - Vol.4 - n.1 - 1.965 - pg.25-30

(Brodie,87)
BRODIE . L.
Starting FORTH
FORTH , Inc - 1987
Prentice Hall, Inc

(Brooms,78)
BROOMS, K.D.
Design of the Fusing System for a Electrophotographic Laser Printer
IBM J. Des. Development - Vol. 22 - n.1 - January 1.978 - pg. 26-33

(Burkett,85)
BURKETT, S.
Smart Laser Printers Merge Text Graphics
Computer Technology Review
Winter 1.985 - pg. 156-157

(Bustamante,84)
BUSTAMANTE, A. et alli
Toner Concentration Control
IBM Technical Discrosure Bulletin - Vol. 26 - n.10A - March 1.984 - pg. 5012-5014

(Cammis,82)
CAMMIS, T.
Laser Printer Image Development System
Hewllet Packard Journal - June 1.982 - pg.20-24

(Casey,82)
Casey, R. et alli
Automatic Scaling of Digital Print Fonts
IBM J. Res. Development - Vol.26 - N.6 - November, 1.982

(Castro,84)
CASTRO, F. AND HANSEN, H.
New Data Cloking Method for a Laser Printhead
IBM Technical Disclosure Bulletin - Vol. 26 - N. 8 - January 1.984 - pg. 4273-4274

(Catalano,85)
CATALANO, F.
Laser Printers Zapping The Impact Market
Electronic Business - May 1, 1985 - pg. 84-90

(Cheung,82)
CHEUNG, L. et alli
Selenium Tellurium Alloys as Photoconductors
Photographic Science and Engineering - Vol. 26 - n.5 - September,December 1.982

(Crow,78)
CROW, F.
The use of grayscale for improved raster display of vectors and characters
Computer Graphics , vol 12 , n 3 , Aug. 1978

(Crumly,82)
CRUMLY, D. AND HANSEN, L.
Laser Printer Machine Control System
Hewllet Packard Journal - July 1.982 - pg.11-15

(Dattilo,74)
DATTILO, ET ALLI
Scanning Light Syncronization System
US Pat 3835249 - 1.974

(Derick,83)
DERICK,M. et alli
Forth Enciclopedia
Mountain View Press, 1983

(Dickison,85)
DICKISON. J.
Laser Printers
PC Magazine - pg.208-216 - September 17, 1.985

(Douglas,87)
Strip-Buffer VS Full-Page Bit-Map Imaging
BYTE - September, 1.987 - pg.229-230

(Elzinga,81)
ELZINGA , C.D. ET ALLI
Laser Electrophotographic Printing Technology
IBM J. Res. Development - Vol. 25 - n.5 - September 1.981

(Enns,86)
ENNUS, S.
Free-Form Curves On Your Micro
BYTE, December,1.986, M 225-230

(Findley,78)
FINDLEY, G. et alli
Control of the IBM 3800 Printing Subsystem
IBM J. Res. Development - Vol. 22 - N.1 - January 1.978 - pg.2-12

(Fitzgerald,82)
FITZGERALD, K.
Interactive Software for Intelligent Printers
Hewllet Packard Journal - June 1.982 - pg. 10-16

(Fleischer,73)
FLEISCHER, J.
Light Scanning and Printing System
US Pat 3750189 - 1.973

(Fleisher,77)
FLEISCHER, J. et alli
Laser Optical System of the IBM 3800 Printer
IBM J. Res. Development - September 1.977 - pg. 479-483

(Gordon,82)
GORDON, P.
Specialized High-Speed Electronics for Document Preparation Flexibility
Hewllet Packard Journal - June 1.982 - pg. 30-35

(Graf,81)
GRAF, P. AND TLUCK ,W.
Sensoren in Laserdruckern
Siemmens Forch. - Vol. 10 - 1.981 - n.2 - pg. 98-103

(Grammatica,81)
GRAMMATICA, S. AND HORT, J.
Infrared Sensitive Organic Photoconductor
Appl. Phys. Lett. - Vol. 6 - n.15 - March 1981 - pg.445-446

(Grant,79)
GRANT , D.
Optical Mechanical Design of the IBM 6670 Laser Printhead
SPIE - Vol.200 - Laser Recording and Information Handling
1.979 - pg.195-199

(Guttag,86)
GUTTAG, K. et alli
Requirements for a VLSI Graphics Processor
IEEE CG&A - Vol. 6 - n.1 - January 86

(Guttag,88)
GUTTAG, K. et alli
The TMS34010, An Enbedded Microprocessor
IEEE Computer Graphics and Aplications - June, 1.988 -
pag.39-52

(Hall,82)
HALL, J.
Laser Printing System Provides Flexible, High Quality, Cost
Effective Computer Output
Hewllet Packard Journal - June 1.982 - pg. 3-8

(Harpavat,79)
HARPAVAT, G.
A Theoretical Study of the Mechanics of a Xerographic
Cleaning Blade
IEEE Transactions on Industry Applications - Vol. IA-15 -
n.6 - November, December 1.979 - pg. 681-687

(Hayashida,83)
HAYASHIDA, B.
F.O.Lens System of Four Group Construction
US Pat 4400063 - 1.983

(Iimura,83)
IIMURA, T. et alli
Ferrite Carrier for Electro-Photographie
IEEE Transactions on Magnetics-Vol. Mag.19 - n.5 - September 1.983 - pg. 1781-1783

(Jurgen,87)
JURGEN, R.
What It Can an Can Not Do-Desktop Publishing
Computer Design - March 1.987 - pg. 50-52

(Juve,82)
JUVE , R. AND RONALD, D.
Monitoring the Laser Printing Process
Hewllet Packard Journal - June 1.982 - pg.26-30

(Karan,82)
KARAM, R.E. AND FARIA, S.
Electrophotographic Versatility of Cadminium Sulfide Type Photoconductors
Journal od Applied Photographic Engineering - Vol. 8 - n.6 - December 1982 - pg.245-250

(Knuth,86)
KNUTH, D.E.
Computer Modern Typefaces
Computers & Typesetting - Vol. E - Addison Wesley - 1.986

(Langley,82)
LANGLEY, J.
Laser Printing System Architecture
Hewllet Packard Journal - June 1.982 - pg.8-10

(Lee,84)
LEE ,M. ET ALLI
Technology Trends in Electrophotography
IBM J. Des. Development - Vol. 28 - n.3 - May 1.984 - pg. 241-251

(Lehmbeck,79)
LEHMBECK, D. et alli
Optical Principles and Practical Considerations for Reflection Microdensitometry
Journal of Applied Photographic Engineering - Vol.5 - n.2 - Spring 1.979 - pg. 63-74

(Lewis,82))
LEWIS , J. AND HUBBY, L.
Optical System Design for the Laser Printing System
Hewllet Packard Journal - July 1.982 - pg. 3-10

(Lieborman,84)
LIEBERMAN, D.
Nonimpact Printers Broader Market Base
Electronic Products - December 12, 1.984 - pg.47-56

(Loelinger,81)
LOELINGER, R.G.
Threaded Interpretive Languages
Mac Graw Hill Book Company/ Byte Books 1.981

(Lutus,81)
Transforth II
Insoft Co. - 1.981

(Maeda,84)
MAEDA, H.
An F.O. Lens System
US Pat 4436383 - 1.984

(Mayer,87)
MAYER, J.
Price Drop Triggers Boom in Laser Printers
Computer Design - January 1, 1987 - pg. 96-100

(McMurtry,84)
MC MURTRY et alii
Technology of the IBM 3800 Printing Subsystem Model 3
IBM J. Res. Development - Vol. 28 - n.3 - May 1.984 - pg.257-262

(Merrit,85)
MERRIT, R.
Laser Printers Zap Graphics Costs
I&CS The Industrial and Process Control Magazine - pg. 71-74
- April 1985

(Meye,77)
MEYE, W.
Optical Character Generation for a High Speed Nonimpact Printer
The Journal of Photographic Science - Vol. 25 - 1977 - pg.183-186

(Mikami,82)
MIKAMI, T. et alli
A Correction Method for Laser Scanning Errors in High Speed Laser Printers
Fujtisu Sci. Tech. J. - Vol. 18 - n. 4 - pg.579-594 December 1982

(Miller,84)
MILLER, R.C.
Introduction to the IBM 3800 Printing Subsystem Models 3 and 8
IBM J: Res. Development - Vol. 28 - n.3 - May 1.984 - pg.254-256

(Minor,82)
MINOR, J.
Hot Roll Fuser
US Pat 4357388 - 1.982

(Myers,84)
MYERS, R.A. AND TAMULIS, J.C.
Introduction to Topical Issue on Non-Impact Printing Technologies
IBM J: Res. Development - Vol. 28 - n.3 - May 1.984 - pg.234-240

(Nakai,82)
NAKAI, A. et alli
Paper-Loop Control at the Fuser Station of a Laser Printer
Journal of Applied Photographic Engineering - Vol. 8 - pg. 167-171 - 1.982

(Nakayama,82)
NAKAYAMA, Y. et alli
A. New a-Si:H Photoreceptor Drum
Preparation, Eletrophotographic Properties and Application
Photographic Science and Engineering - Vol.26 - n.4 - July/August 1982 - pg.188-193

(Nebenzahl,80)
NEBENZAHL , L. et alli
Forces Involved in Cleaning of an Electrophotographic Layer
Photographic Science and Engineering - Vol.24 - n.6 - November, December 1.980

(Nelson,78)
NELSON, K.
Electrographic Development Process
US Pat 4121931 - 1.978

(Newkirk,83)
NEWKIRK, J.
Fuser Member
US Pat 4375505 - 1.983

(Neuman,81)
NEWMAN, W and SPROULL, R
Principles of Interactive Computer Graphics
Second Edition - 1981
McGraw Hill Book Company

(Ohr,84)
OHR, S.
Intelligence Helps Nonimpact Printers Make their Mark
Electronic Design - June 28, 1.984 - pg. 144-156

(Pavlidis,79)
PAVLIDIS, THEO
Filling Algorithms for Raster Graphics
Computer Graphics and Image Processing - Vol.10 - n.2 - June, 1.979 - pg.126-141

(Pavlidis,81)
PAVLIDIS, THEO
Filling Algorithms for Raster Graphics
Computer Graphics and Image Processing - Vol.15 - n.13 - June, 1.981 - pg.29-36

(Pike,83)
PIKE, R.
Graphics in Overlapping Bitmap Layers
ACM Transactions on Graphics - Vol. 2 - n.2 - April 1983 - pg. 135-160

(Pitteway,67)
PITTEWAY, M.L.
Algorithm for Drawing Ellipses or Hyperboloe with a Digital Plotter
Computer J. - Vol. 10 - n.3 - November, 1.967 - pg.282-289

(Porter,84)
PORTER, T. AND DUFF, T.
Compositing Digital Images
ACM - Computer Graphics - Vol. 18 - n.3 - July 1984 - pg. 253-259

(Pountain,87)
POUNTAIN , D.
Vector Raster Algotithms
Byte, September 1987 - pg. 177-184

(Prime,83)
PRIME, B.
Relationships Between Toner Properties, Fuser Parameters and Fixing of Electrophotographic Images
Photographic Science and Engineering - Vol. 27 - pg. 19-25 - 1.983

(Rabedeau,78)
RABEDEAU, M.
Optical System for Rotating Mirror Line Scanning Apparatus
US Pat 4123135 - 1978

(Rogers,85)
ROGERS, D
Procedural Elements for Computer Graphics
1985
McGraw-Hill Book Company

(Rosemberg,85)
ROSEMBERG, R.
Quiet Print Invades Office
Electronics Week - March 18, 1985 - pg. 59-63

(Rothgordt,82)
ROTHGORDT, ULF.
A Survey of Electronic Printing Technologies
Advances in Image Pickup and Display - Vol. 5 - 1.982
Academic Press Inc.

(Russel,84)
RUSSEL, M. AND BAUGHMON, J.
Raster Image Processing Provides the Driver for Nonimpact Devices
Computer Technology Review - Summer 1.984 - pg.165-171

(Salesin,86)
SALESIN, D. AND BARZEL, R.
Two Bit Graphics
IEEE CG&A - Vol. 6 - n.6 - June 1.986 - pg. 36-42

(Schaffert,71)
SCHAFFERT, R.M.
A New High Sensitivity Organic Photoconductor for Electrophotographie.
IBM J. Res. Development - January 1971 - pg. 75-89

(Schaffert,78)
SCHAFFERT, R.M.
Electrophotographie Yesterday, Today and Tomorrow
Photographic Science and Engineering - Vol. 22 - n.3 - May,June 1.978

(Schein,75)
SCHEIN, L.B.
Microscopic Theory of Magnetic Brush Development
Photographic Science and Engineering - Vol.19 - n.5 - September/ October 1.975 - pg.255-265

(Schwiebert,82)
SCHWIEBERT, E. et alli
Electrostatic Image Formation in a Laser Printer
Hewllet Packard Journal - June 1.982 - pg.16-20

(Shibuya,83)
SHIBUYA, M.
F.O. Lens
US Pat 4396254 - 1.983

(Shimada,88)
SHIMADA, et alli
Scanning Clock Generating Device for Optical Scanner
US Pat 4729617 - 1.988

(Smith,79)
SMITH, A.R.
Tint Fill
Computer Graphics - Vol.13 - n.02 - August, 1.979 - pag.276-283

(Starkweather,80)
STARKWEATHER, G.K.
High-Speed Laser Printing Systems
Laser Applications - Vol. 1 - 1.980 - Academic Press Inc.

(Stefani.85)
STEFANI , M. A.
Sistema de Gravação Laser
CEMEFI-EESC-USP - 1985
Relatorio final do Projeto "Nacionalização de uma Maquina Gravadora a Laser" - FIPEC - Banco do Brasil

(Stefani,86)
STEFANI , M. A. et alli
Estudo de Viabilidade de Nacionalização de uma Impressora Laser
CEMEFI-EESC-USP - 1986
Relatório final do Projeto de mesmo nome , apresentado a SEI-CTI, financiado pela firma Expansão Informatica LTDA.

(Stefani,88)
STEFANI , M. A. et alli
Relatório de Andamento de Projeto , período 1987-1988.
Desenvolvimento de uma Impressora Laser
Relatório apresentado ao FINEP - 1988

(Svrendsen,78)
SVRENDSEN,.
Paper Path of an On-line Computer Output Printer
IBM J. Res. Development - Vol. 22 - N.1 - January 1.978 - pg.13-18

(Tam,82)
TAM, A. et alli
Lasers in Electrophotography
IBM J. Des. Develop.-Vol.26- n.2 - March 1.982 - pg.186-197

(Takahashi,82)
TAKAHASHI, T. et alli
Mechanism of Canon Toner Projection Development
Photographic Science and Engineering- Vol. 26 - n.5
September/October 1982 - pg.254-261

(Tateoka,79)
TATEOKA, et alli
Device for Scanning a Light Beam at a Constant Speed
Us Pat 4179183 - 1.979

(Teja,85)
TEJA, ED.
Printers
EDN - February 7 , 1.985 - pg. 157-172

(Texas,86)
TEXAS INSTRUMENTS
TMS 34010 User Guide , 1.986

(Texas-1,87)
TEXAS INSTRUMENTS
Software Development Board User Guide , 1.987

(Texas-2,87)
TEXAS INSTRUMENTS
TMS 34010 Assembly Language Tools User Guide , 1.987

(Texas,88)
TEXAS INSTRUMENTS
TMS 34020/34082 Graphics Products Preview Bulletin SPUT065
1.988

(Thoma,81)
THOMA, W.G.
Offset - Teoria e Aplicações
Projeto Editora Associados Ltda - 1.981

(Ting,86)
TING, C.H.
Inside F83
Offete Enterprises , Inc. 1.986

(Tu,75)
TU, Y.O.
Theory of Liquid Ink Development in Electrophotographie
IBM J. Res. Development - November 1.875 - pg. 514-522

(Vahtra,78)
VAHTRA, V. AND WOLTER, R.
Eletrophotographic Process in a High Speed Printer
IBM J. Res. Development - Vol.22 - n.1 - January 1.978 -
pg.34-39

(VanAken,84)
VANAKEN, J.
An Efficient Ellipses - Drawing Algorithm
IEEE Computer Graphics and Aplications - September, 1.984 - pg.24-35

(Yamakawa,89)
YAMAKAWA, K.
Tilt Error Corrective Scanning Optical System
US Pat 4799747 - 1.989

(Warnock,82)
WARNOCK, J. AND WYATT, D.
A Device Independent Graphics Imaging Model for use with Raster Devices
ACM Computer Graphics - Vol. 16 - n.3 - July 1982 - pg.313-319

(Willians,85)
WILLIANS, T.
Options Multiply for Nonimpact Page Printers
Computer Design , February 1985 - pg. 33-38

(Wilson,79)
WILSON ,C.C.
A New Fuser Technology for Electrophotographic Printing Machines
Journal of Applied Photographic Engineering - Vol. 5 - Number 3 - Summer 1.979

(Wolter,78)
WOLTER, R.F.
The IBM 3800-Electrophotographie in Computer Output Printing
Journal of Applied Photographic Engineering - Vol. 4 - n.4 - 1.978

APENDICE I) RELAÇÃO DOS COMANDOS IMPLEMENTADOS

AI-1) Introdução

Será apresentado a seguir a relação completa das funções e comandos implementados na linguagem. Muitos dos comandos seguem o padrão encontrado na linguagem FORTH , permitindo assim a melhor compreenção de suas características, atraves de exemplos didáticos encontrados na literatura.

Para a descrição será usada a convenção de símbolos seguir:

a) Atributos / Vocabulários:

R	palavra pertencente ao vocabulário CORE , raiz
C	palavra pertencente ao vocabulário COMPILER, somente permitido seu uso no modo compilativo
G	palavra pertencente ao vocabulário GRAPHICS
I	palavra com o atributo "imediato" , sendo executado mesmo no modo compilativo.
DD	palavra cujo código depende da implementação "device dependent".

b) Pilhas:

- as entradas e saídas na pilha de parâmetros são mostradas, como a seguir , sendo o símbolo "¦" o representante da função sendo descrita

(entradas ¦ saídas)

- dentre os vários elementos na entrada (ou saída) , o mais a direita está no topo, ou foi o último a ser colocado

- caso não haja uma entrada (ou saída), o sinal "-" é utilizado

- caso se represente outra pilha , além da de parâmetros , são usadas as abreviações

R(entradas ¦ saídas) pilha de retorno
I(entradas ¦ saídas) pilha de índices

- entre chaves { } são colocados comentários

- entre chaves e aspas {" "} é colocada a "pronuncia" dos comandos, segundo a convenção da linguagem FORTH-83

- como entradas e saídas são usados os seguintes símbolos , representantes dos tipos de dados usados e gerados pela função . Obs: a pilha sempre armazena dados em 32 bits.

flag	sinalizador booleano , TRUE (-1) ou FALSE (0) (esporádicamente pode assumir o valor 1)
char	caracter , ASCII
string	conjunto de caracteres .
x,y,z	números inteiros , com sinal (32 bits, compl. 2) (-2147483648 < x < 2147483648)
idx	número inteiro , índice de um laço DO-LOOP
lmt	número inteiro , limite final de um índice em um laço DO-LOOP
quot	número inteiro , com sinal , resultado de uma divisão inteira (quociente)
mod	número inteiro , com sinal , resto de uma divisão inteira (modulus)
u	número inteiro , sem sinal (32 bits) (0 < u < 4292967296)
w	número de 16 bits , normalmente os "lsb" de um número de 32 bits na pilha
b	número de 8 bits , similar a w
addr	endereço , 32 bits , similar a um número inteiro sem sinal
adf	endereço de execução de uma função , primeira célula executável do corpo de uma definição
adh	endereço do cabeçalho de uma função, contendo o elo com a definição anterior no mesmo vocabulário.
adpf	endereço do "parameter field" de uma função. E' o endereço em variáveis , constantes, etc , onde serão armazenados os dados . Geralmente é o endereço da primeira célula livre, também chamado de campo de parâmetros.
source	endereço do "parameter field" ou corpo de um stencil, que será usado como origem em uma ação gráfica entre stencils.
dest	endereço de "parameter field" ou corpo de um stencil, que será usado como destino em uma ação gráfica entre stencils.

ppop "pixel processing option" . E um número código que especifica a opção de processamento gráfico que ocorrerá em ações entre stencils . Trata-se de um código usado para fixar variáveis internas do processador TMS34010.

pitch é a diferenca de endereços na memória gráfica entre dois pixels de mesma coluna , separados de uma linha.

c) Outros abreviações e símbolos:

string conjunto de caracteres que segue um comando

name conjunto de caracteres que segue um comando , que será usado para nomear alguma entidade

lsb "loss significant bit" , ou os bits mais a direita de um número binário

msb "most significant bit" , ou os bits mais a esquerda de um número binário

d) Significado de alguns termos:

- "endereço de retorno" de uma função é o endereço que a função deve retornar quando terminar ou encontrar sua instrução interna RETS. Este endereço está no topo da pilha de retorno quando se iniciou sua execução.

-"célula" : espaço de 32 bits , no dicionário

-"meia célula" : espaço de 16 bits , no dicionário

-"célula de retorno" de uma função é o espaço de 32 bits , no dicionário , localizado no endereço de retorno da função em questão .

-"vocabulário de contexto" é aquele onde as buscas são feitas prioritariamente

-"vocabulario corrente" e aquele onde as novas definições serão colocadas

-"acumuladores" ou "buffers" são regiões de memória destinadas a acumular dados visando operações de entrada/saída.

-"acumulador de palavra" ou "word buffer" é uma pequena região livre no dicionário , a frente de HERE , destinada a ser área de rascunho no processamento de "strings"

-"compile-time" , é o comportamento que uma função tem ao ser compilada, ou no modo de compilação.

-"run-time" , é o comportamento que uma função tem ao ser executada , ou no modo interpretativo.

-"stencil" , é a entidade gráfica que representa uma área bidimensional contendo uma imagem a ser trabalhada ou impressa.

-"trajetória" , é uma entidade gráfica que representa a expressão analítica que uma figura ou forma possui. Representa o caminho ou trajetória que um elemento "pintante" vai percorrer para produzir a forma.

-"ponto corrente", ou "current point" é o registro da coordenada de um ponto selecionado dentro de um stencil.

-"brush" , ou "pincel" é um stencil que será usado para simular o efeito de um pincel sendo escorregado pelo papel , seguindo uma trajetória.

-"stencil corrente" é um stencil pre-selecionado , onde serão aplicadas as trajetórias escolhidas e executadas a seguir.

-"fonte" é um conjunto de caracteres definidos usando um mesmo estilo tipografico. Pode ser constituido de dois tipos: caracteres definidos por trajetórias ou definidos por stencils.

-"encoding vector" , ou matriz codificadora , é uma matriz que organiza a localização das diversas definições de caracteres que compõe um "fonte"

-"memória gráfica" , é a região da memória destinada a conter as imagens que serão impressas.

-"pixel" é um ponto na memória gráfica , contendo a informação "branco" ou "preto"

A!-2) Relação das funções

- Funções de manipulação da pilha:

DUP (x | x x) R
duplica o topo da pilha

DROP (x | -) R
elimina o topo da pilha

SWAP (x y | y x) R
intercambia os elementos no topo

OVER (x y | x y x) R
copia o segundo item no topo

ROT (x y z | y z x) R
coloca o terceiro item no topo

R> (- | x) C {"r-from"}
R(x | -)
Transfere o número do topo da pilha de retorno para o topo da pilha de parâmetros

>R (x | -) C {"to-r"}
Transfere o número do topo da pilha para o topo da pilha de retorno.

R@ (- | x) C {"r-fetch"}
Copia o topo da pilha de retorno na pilha de parâmetros

DDUP (y x | y x y x) G {"double-dup"}
Duplica os dois elementos no topo da pilha.

DROPTRUE (TRUE x x .... | -) G
Elimina os elementos no topo da pilha até encontrar o sinalizador TRUE.

- Operadores lógicos:

NOT (x | y) R
complemento de um

AND (x y | z) R
função lógica "and"

OR (x y | z) R
função lógica "or"

XOR (x y | z) R {"x-or"}
função lógica "exclusive-or"

- Funções de comparação:

< (x y | flag) R {"less-than"}
flag = TRUE se x<y , FLASE caso contrário

> (x y | flag) R {"greater-than"}
flag = TRUE se x>y , FALSE caso contrário

= (x y | flag) R {"equal"}
flag = TRUE se x=y , FALSE caso contrário

0< (x | flag) R {"zero-less"}
flag = TRUE se x<0 , FALSE caso contrário

0> (x | flag) R {"zero-greater"}
flag = TRUE se x>0 , FALSE caso contrário

0= (x | flag) R {"zero-equal"}
flag = TRUE se x=0 , FALSE caso contrário

0<> (x | flag) R {"non-zero"}
flag = TRUE se x diferente de zero, False caso contrário.

U< (u1 u2 | flag) R {"unsigned-less"}
Comparação de dois números sem sinal, TRUE se u1<u2

- Funções aritméticas:

+ (x y | z) R {"plus"}
soma dois números , com sinal

- (x y | z) R {"minus"}
subtrai dois números , com sinal (z=x-y)

* (x y | z) R {"times"}
multiplica dois números , com sinal

/ (x y | quot) R {"divide"}
divide x/y , apresentando o quociente

MOD (x y | mod) R
divide x/y , apresentando somente o resto

/MOD (x y ¦ mod quot) R ("divide-mod")
divide x/y , apresentando quociente e resto

*/MOD (x y z ¦ mod quot) R ("times-divide-mod")
multiplica x*y e depois divide por z, usando 64 bits na operação intermediária

*/ (x y z ¦ quot) R ("times-divide")
similar a */MOD , sem apresentar o resto

MAX (x y ¦ z) R
z é o maior elemento entre x e y , considerando sinal

MIN (x y ¦ z) R
z é o menor elemento entre x e y , considerando sinal

ABS (x ¦ z) R ("absolute")
valor absoluto de x

NEGATE (x ¦ z) R
complemento de dois do número x

+! (x addr ¦ -) R ("plus-store")
soma x ao conteúdo de addr

ADDXY (y1 x1 y2 x2 ¦ y3 x3) G
Função auxiliar , somando as coordenadas de dois pontos y,x.

y3 = y2 + y1
x3 = x2 + x1

- Funções de manipulação de strings:

HOLD (char ¦ -) R
Insere o caracter no string sendo montado em PAD, incrementando HLD

<# (- ¦ -) R ("less-sharp")
inicia a conversão de um número em string, preparando PAD.

(x ¦ y) R ("sharp")
converte um digito do número na pilha , coloca o caracter correspondente em PAD e o número restante de volta na pilha

SIGN (x ¦ -) R
se o número for negativo insere um "-" no string sendo montado em PAD.

#> (x | addr n) R {"sharp-greater"}
termina a conversão de um número em string. Coloca na pilha o endereço e o comprimento do string recém-formado.

#S (x | y) R {"sharp-s"}
repete a função " # " até que o número restante seja zero.

(.) (x | addr) R {"paren-dot"}
converte um número no string , até seu fim , incluindo o sinal , e deixa seu comprimento na pilha. O comprimento está no primeiro byte do string.

. (x | -) R {"dot"}
pega o número, converte para string e envia para o acumulador de saída .

- Funções de acesso a memória:

@ (addr | x) R {"fetch"}
recupera o número contido no endereço

B@ (addr | b) G {"b-fetch"}
recupera o número de 8 bits , lsb , contido no endereço

! (x addr | -) R {"store"}
armazena o número no endereço

C@ (addr | w) R {"c-fetch"}
recupera o número (16 bits,lsb) contido no endereço

C! (w addr | -) R {"c-store"}
armazena o número (16 bits,lsb) no endereço

D@ (addr | y x) G {"double-fetch"}
recupera o número contido no endereço addr , correspondendo a "y" , é o número contido no endereço addr+32 , correspondente a "x". Usado para recuperar valores em variáveis duplas criadas por DOUBLEVAR

D! (y x addr | -) G {"double-store"}
Armazena o número "y" em addr , e "x" em addr+32 . Usado para armazenar valores em variáveis duplas criadas por DOUBLEVAR

- Estruturas de controle:

IF-ELSE-THEN C !

Usado das seguintes formas no "run time":

```
( flag | - )
     IF  (executa se flag=TRUE)  THEN
ou
     IF  (executa se flag=TRUE)  ELSE
         (executa se flag=FALSE) THEN
```

No "compile time"

```
IF   ( - | addr )
     monta o comando ?BRANCH no dicionário ,  e executa
     o comando >MARK .
ELSE ( addr1 | addr2 )
     monta o comando BRANCH no dicionário ,  executa os
     comandos >MARK , SWAP e >RESOLVE
THEN ( addr | - )
     executa o comando >RESOLVE
```

BEGIN-UNTIL C !

Usado da seguinte forma no "run-time":

```
BEGIN  ( repete se flag=FALSE ) flag UNTIL
       ( sai se flag=false )
```

No modo compilativo:

```
BEGIN ( - | addr )
     executa o comando <MARK
UNTIL ( addr | - )
     monta  o comando ?BRANCH no dicionário e executa o
     comando <RESOLVE
```

BEGIN-AGAIN C !

Usado da seguinte forma no "run-time":

```
BEGIN  ( repete indefinidadmente ) AGAIN
```

executa um laço infinito.

No modo compilativo:

```
BEGIN ( - | addr )
     executa o comando <MARK
AGAIN ( addr | - )
     monta  o comando BRANCH no dicionário e executa  o
     comando <RESOLVE
```

BEGIN-WHILE-REPEAT C !

Usado da seguinte forma no "run-time":

```
BEGIN  ( trecho 1 ) flag WHILE (trecho 2 ) REPEAT
     Repete  trecho 1 e trecho 2  se  flag=FALSE,  pula
     trecho 2 e sai se flag=TRUE
```

No modo compilativo:

BEGIN (- | addr)
executa o comando <MARK
WHILE (- | addr)
monta o comando ?BRANCH no dicionário e executa o comando >MARK
REPEAT (addr1 | addr2)
monta o comando BRANCH no dicionário , executa os comandos SWAP , <RESOLVE e >RESOLVE.

DO-LOOP
DO-+LOOP

C !

Usados da seguinte forma no "run-time"

(lim idx | -) DO (trecho 1) LOOP
repete o trecho 1 , de "idx" até "lim" , incremento 1.
(lim idx | -) DO (trecho 1) inc +LOOP
repete o trecho 1 , de "idx" ate "lim" , incremento "inc".
Na execução , durante o trecho 1 , o "lim" e o "idx" são colocados na pilha de índices , bem como o endereço de saída.

No "compile time":
DO (- | addr)
monta no dicionário a função RDO e executa >MARK
LOOP (addr | -)
monta no dicionário a função RLOOP , soma 32 ao "addr" e executa <RESOLVE e >RESOLVE.
+LOOP (addr | -)
idem a LOOP , montando a função RPLUSLOOP.

LEAVE

C !

Usado da seguinte forma no "run-time":

(ind lim | -) DO (trecho 1) LEAVE (trecho 2) LOOP
Se LEAVE é executado , cancela execução do laço DO-LOOP, saltando sobre o trecho 2.

No "compile time":
LEAVE (- | -)
monta a função RLEAVE no dicionário

?LEAVE

C !

Usado da seguinte forma no "run-time":

(ind lim | -) DO (trecho 1) flag ?LEAVE
(trecho 2) LOOP
Se flag= TRUE , cancela execução do laço DO-LOOP, saltando sobre o trecho 2.
No "compile time":
?LEAVE (- | -)
monta a função RCLEAVE no dicionário

I (- ¦ idx) C
copia na pilha o índice corrente do primeiro laço DO-LOOP mais interior

J (- ¦ idx) C
copia na pilha o índice corrente do segundo laço DO-LOOP mais interior

K (- ¦ idx) C
idem para o terceiro laço

- Funções para entrada/saída de dados:

TYPE (addr ¦ -) R
Transmite ao acumulador de saída TOB o "string" iniciado no endereço. O primeiro Byte é considerado como Comprimento. Aciona OUTPUT .

TOKEN (char ¦ -) R
Retira do acumulador de entrada TIB, a próxima palavra, delimitada pelo caracter fornecido na pilha. Coloca no acumulador de palavra.

BASE (- ¦ adpf) R
Variável de controle que armazena a base de conversão numérica adotada.

HLD (- ¦ adpf) R
Variável de controle que armazena o ponteiro no acumulador de rascunho PAD

PAD (- ¦ adpf) R
Variável de controle que armazena o endereço inicial de uma área de rascunho, situada a 256 bytes de HERE , para conversões numéricas

INPUT (- ¦ -) R DD
Rotina de interfaceamento com o canal de comunicação. Pega os comandos enviados ao sistema e armazena no TIB até receber um caracter "carriage return". Aciona os sinalizadores ENDBUF e o ponteiro >IN correspondentemente.

OUTPUT (- ¦ -) R DD
Rotina de interfaceamento com o canal de comunicação. Pega os dados constantes em TOB e os envia para o Interface de comunição , se esta permitir.

TIB (- ¦ adpf) R
Variável de controle que mantém o endereço do "text input buffer" , ou acumulador de dados de entrada . Situado na posição HERE+256.

TOB (- ¦ adpf) R
Variável de controle que mantém o endereço do "text output buffer" , ou acumulador de dados de saída. Situado na posição HERE+512.

NENDBUF (- ¦ adpf) R
Variável de controle que mantém um sinalizador indicando se o acumulador TIB está vazio.Se TRUE, vazio Se FALSE, ainda existem comandos.

>IN (- ¦ adpf) R {"to-in"}
Variável de controle . Mantém um ponteiro indicando o caracter corrente dentro de TIB.

TOBP (- ¦ adpf) R
Variável de controle , Mantém um ponteiro indicando o caracter corrente no TOB , quando este existir.

ABORT" string " (flag ¦ -) C I {"abort-quote"}
Se o sinalizador for TRUE , envia a mensagem para TIB e aborta o sistema.

- Funções de acesso ao dicionário:

HERE (- ¦ addr) R
retorna na pilha o endereço da próxima posição livre no dicionário

DIP (- ¦ adpf) R
retorna na pilha o endereço do pointer do dicionário

- Funções do compilador/interpretador:

EXECUTE (adf ¦ -) R
executa a função cujo endereço de execução está no topo da pilha. A função deve terminar com a intrução "RETS"

ALLOT (u ¦ -) R
reserva "u" células no dicionário.

, (n ¦ -) R {"comma"}
copia topo da pilha no dicionário

C, (w ¦ -) R {"c-comma"}
copia os 16 lsb do topo da pilha na próxima meia-célula do dicionário

," string " R ("comma-quote")
compila o string no dicionário .

." string " C ! ("dot-quote")
compila o string no dicionário , inserindo a função " (.") " ("paren-dot-quote") na célula inicial.

" string " C ! ("quote")
compila o string no dicionário , inserindo a função " (") " ("paren-quote") na célula inicial.

ASCII char C !
compila o próximo caracter no dicionário . No "run-time" coloca na pilha.

LITERAL (x | -) C !
Insere a função LIT no dicionário e pega o número que está na pilha e o compila na célula seguinte.

COMPILE name C
(- | -)
Ao se executar a definição que contém este comando, coloca no dicionário uma chamada à função cujo endereço está na célula de retorno (dentro de uma definição insere um CALLA name).

[COMPILE] name C ! ("bracket-compile")
(- | -)
Idem a COMPILE , incluindo as funções imediatas

['] name C ! ("bracket-tick")
Compila o endereço da próxima palavra que está na linha de comando no TIB , no dicionário como um literal. No "run-time", coloca na pilha.

CREATE (- | -) R
Cria, no dicionário, um cabeçalho com a próxima palavra a ser encontrada no acumulador de entrada, TIB.

MODE (- | flag) R
Se flag = TRUE o sistema está no modo compilativo
Se flag = FALSE o sistema está no modo interpretativo

<BUILD (- ¦ adf)
DOES> (adf ¦ -) C !

Comandos usados para delimitar regiões de comportamento diferenciado dentro de uma função geradora de novas estruturas. Usado da seguinte forma :

: father <BUILDS [trecho1] DOES> [trecho2] ;

A função "father" é um novo gerador de estrutura definido pelo usuário. Ao "father1" ser executado , cria a nova estrutura "child" , para isto basta:

father child

Neste momento o trecho1 é o utilizado.
Quando "child" for executado , a execução desvia para o trecho2 de "father"
<BUILDS gera um cabeçalho com a próxima palavra no TIB e insere no adf uma instrução CALLA. O endereço da próxima célula é colocado na pilha
DOES> pega o endereço da sua célula de retorno e armazena este endereço na célula cujo endereço está na pilha

[(- ¦ -) C ! ("left-bracket")

Interrompe o modo de compilação, ativa interpretação .

] (- ¦ -) C ("right-bracket")

Retorna ao modo de compilação

' name (- ¦ adf) R ("tick")

coloca na pilha o endereço de execução da palavra seguinte que está na linha de comando no TIB .

COMPILATOR (adf flag ¦ -) R

Rotina de controle do modo compilativo. Pega o "adf" e o flag deixado por DEFINED e :
-se flag=TRUE , insere a intrução CALLA e compila o "adf" na próxima célula
-se flag=1 , executa o endereço "adf"
-se flag=FALSE ,"adf" é o endereço do "word buffer", Nesta situação, tenta transformar a palavra num número através de NUMBER e insere o comando LIT seguido do número na próxima célula. Se a transformação não for bem sucedida , imprime mensagem de erro e aborta.

INTERPRETER (adf flag ¦ -) R

Rotina de controle do modo interpretativo . Pega o "adf" e o flag deixado por DEFINED e:
-se flag=TRUE ou flag=1 , executa o endereço "adf"
-se flag=FALSE , "adf" é o endereço do "word buffer". Nesta situação, tenta transformar a palavra num número através de NUMBER e coloca o correspondente número na pilha. Se a transformação não for bem sucedida , imprime mensagem de erro e aborta.

INNER R

Rotina de controle do sistema. Acionado por ABORT. Consta em um laço infinito , que mantém o sistema em funcionamento . Primeiramente aciona INPUT, depois pega cada comando através de TOKEN , pesquisa no dicionário através de DEFINED e aciona COMPILATOR ou INTERPRETER, de acordo com o modo corrente. Ao esgotar TIB, reaciona INPUT.

- Vocabulários:

CONTEXT (- ¦ adpf) R

coloca na pilha o endereço da variável que contém o vocabulário de contexto. Armazena o endereço da variável que contém o "adh" ,da última palavra definida neste vocabulário.

CURRENT (- ¦ adpf) R

idem para o vocabulário corrente

ENTRY (- ¦ adh) R

retorna o endereço do cabeçalho da última palavra definida no vocabulário corrente

COMPILER (- ¦ adpf) R

retorna o endereço da variável que armazena o endereço do cabeçalho da ultima palavra, definida no vocabulário de compilação

CORE (- ¦ adpf) R

idem para vocabulário raiz

GRAPHICS (- ¦ -) R

Aciona o vocabulário gráfico , colocando em CONTEXT o endereço do cabeçalho da última palavra definida neste vocabulário.

LAST (- ¦ adpf) R

retorna o endereço da variável que contém o endereço do cabeçalho da última palavra criada no dicionário.

COMPILERWORD (- ¦ -) R

Transfere a última palavra definida no vocabulário corrente para o vocabulário de compilação

COREWORD (- ¦ -) R

Idem para o vocabulário raiz

IMMEDIATE (- ¦ -) R

Coloca no cabeçalho da última definição criada no dicionário o atributo de "imediata" . Aciona o bit de precedência.

DEFINED (- ¦ adf flag) R

Rotina de busca de palavras nos vocabulários. Pega a palavra no "word buffer" e percorre o vocabulário de contexto, a seguir o vocabulário corrente e , se no modo de compilação , o vocabulário de compilação.
-se a palavra é encontrada , coloca o seu "adf" e o flag=true,
-se a palavra é encontrada e tem o atributo "imediato",coloca o seu "adf" e o flag=-1,
-se a palavra não é encontrada , coloca o endereço do "word buffer" e o flag=false.

- Funções geradoras de estruturas:

: name (- ¦ -) R ! ("colon")

ativa modo de compilação , cria um cabeçalho da nova função "name" na próxima posição livre do dicionário , encadeia no vocabulário corrente, fazendo com que o vocabulário de contexto seja igual ao corrente.

; (- ¦ -) C ! ("semi-colon")

termina a compilação da função "name" , iniciada pelo último comando : "colon"

VARIABLE name (- ¦ -) R

cria no dicionário a variável "name" , no vocabulário corrente
Na execução de "name" retorna o endereço de se "parameter field"

name (- ¦ adpf)

DOUBLEVAR name (- ¦ -) G

Cria no dicionário a variável dupla "name" , no vocabulário corrente. Este tipo de variável reserva duas células , visando armazenar duplas y,x. No momento da criação , inicializa y=0 e x=1.
Na execução de "name" , retorna o endereço de seu "parameter field" , correspondendo a posição que armazena "y".

ARRAY name (n ¦ -) G

Cria no dicionário a matriz unidimensional de "n" elementos com o nome "name", no vocabulário corrente.
Na execução de "name":

name (i ¦ adpf)

retorna o endereço do i-ésimo elemento.

CONSTANT name (x | -) R

cria no dicionário a constante "name" , de valor x.
Na execução de "name" retorna a constante

name (- | x)

VOCABULARY name (- | -) R

cria no dicionário o vocabulário "name" , a partir do vocabulário corrente.
Na execução de "name" , coloca em CURRENT o "adh" da mais nova entrada ocorrida neste vocabulário.

STENCIL name G

(dy dx | -)
Cria o stencil "name" , com dimensões "dy","dx" , a partir da próxima posição livre na memória gráfica. Assume "pitch"="dy" . Cria no dicionário um cabeçalho com o nome "name" contendo as informações do stencil e posiciona o ponto corrente na origem deste.
Ao se executar "name" , retorna na pilha o endereço do campo de parâmetros.

DOSTENCIL name

(pitch dy dx addr | -) G

Cria um stencil com o nome "name" e dimensões fornecidas na pilha. Cria no dicionário um cabeçalho com o nome "name" que ira armazenar os dados correspondentes ao stencil que está sendo criado . O endereço "addr" corresponde a posição inicial do stencil na memória gráfica, .

- Funções definidoras de contexto gráfico:

GFUNCTION (- | addr) G

Variável de controle que contém a opção de função gráfica corrente, válido para os próximos comandos de ação gráfica.

VTRANSP (- | addr) G

Variável de controle que contém um sinalizador indicando se a opção de transparência foi habilitada. Se TRUE , transparência habilitada por TRANSPARENCY.

VORG (- | addr) G

Variável de controle , contém a configuração da posição da origem 0,0 num stencil
- 0,0 origem acima esquerda (assumido na inicialização)
- 1,0 origem acima direita
- 0,1 origem abaixo esquerda
- 1,1 origem abaixo direita

TRANSPARENCY (flag | -) G

Acessa a variável interna do TMS34010 CONTROL , fixando os bits correspondentes as opções de transpararência , origem VORG e tamanho do pixel. Flag=TRUE habilita .

LOADFUNCTION (ppop | -) G

Acessa a variável interna do TMS34010 CONTROL , fixando os bits correspondentes a operação gráfica desejada , cujo código está na pilha. Operação válida para os próximos comandos de ação gráfica.

GORIGIN (- | adpf) G

Variável de controle , contendo o ponteiro na memória gráfica , indicando a próxima posição livre. Usado para fornecer o endereço inicial de um stencil, sendo criado por STENCIL.

GETCPT (source | y x) G

Põe na pilha o ponto corrente do stencil "source"

SETCPT (y x dest | -) G

Põe no ponto corrente do stencil "dest" a coordenada y,x

CURCPT (- | y x) G

Põe na pilha o ponto corrente do stencil corrente.

CURSTENCIL (- | adpf) G

Variável que contém o endereço do stencil corrente

CURBRUSH (- | adpf) G

Variável que contém o endereço do stencil, que será usado como pincel pelo comando LINEB.

BOUNDARY (- | adpf) G

Variável que contém o valor do pixel adotado para a função FILL . Se contém TRUE , pixel="branco" , False = "preto".

ROTATION (- | adpf) G

Variável de controle que contém o ângulo, em graus,em que as trajetórias devem ser rotacionadas, antes de serem colocadas nos stencils . Inicializada em "0".

VCPT (- | adpf) G

Variável auxiliar dupla , que armazena o ultimo ponto y,x fornecido a um comando MOVETO, LINETO ,RMOVETO e RLINETO. Armazena o ponto corrente no sistema virtual de coordenadas

ORIGIN (- | adpf) G

Variável auxiliar dupla , que armazena a origem desejada dentro de um stencil , onde serão aplicadas as trajetórias escritas em coordenadas relativas.

SCALEX (- | adpf) G
Variável auxiliar dupla , que armazena os parâmetros a,b do eixo X que serão usados na transformação SCALE. Inicializada em 1 , 1 .

SCALEY (- | adpf) G
Variável auxiliar dupla , que armazena os parâmetros a,b do eixo Y que serão usados na transformação SCALE. Inicializada em 1 , 1 .

SHEARX (- | adpf) G
Variável auxiliar dupla , que armazena os parâmetros a,b do eixo X que serão usados na transformação SHEAR. Inicializada em 0 , 1 .

SHEARY (- | adpf) G
Variável auxiliar dupla , que armazena os parâmetros a,b do eixo Y que serão usados na transformação SHEAR. Inicializada em 0 , 1 .

GRAPHMODE (- | adpf) G
Variável auxiliar , que armazena o sinalizador que descrimina o modo gráfico que será adotado. Se contém TRUE , DRAWMODE . Se contém FALSE , FILLMODE

DRAWMODE (- | -) G
Coloca o sistema no modo gráfico correspondente , onde as trajetórias usarão o stencil contido em CURBRUSH como elemento simulador de pincel.

FILLMODE (- | -) G
Coloca o sistema no modo gráfico correspondente , onde as trajetórias usarão o elememto armazenado em BOUNDARY, como delimitador de figuras a serem preenchidas por FILL.

- Funções definidoras de operações gráficas:

REPLACE (- | -) G
Especifica "ppop" como 00000 e executa LOADFUNCTION source -> dest

GAND (- | -) G
Especifica "ppop" como 00001 e executa LOADFUNCTION source AND dest -> dest

GANDNOT (- | -) G
Especifica "ppop" como 00010 e executa LOADFUNCTION source AND (NOT dest) -> dest

BLACK (- | -) G
Especifica "ppop" como 00011 e executa LOADFUNCTION "0" -> dest

GORNOT (- ¦ -) G
Especifica "ppop" como 00100 e executa LOADFUNCTION source OR (NOT dest) -> dest

GXNOR (- ¦ -) G
Especifica "ppop" como 00101 e executa LOADFUNCTION source XNOR dest -> dest

GNOTD (- ¦ -) G
Especifica "ppop" como 00110 e executa LOADFUNCTION NOT dest -> dest

GNOR (- ¦ -) G
Especifica "ppop" como 00111 e executa LOADFUNCTION source NOR dest -> dest

GOR (- ¦ -) G
Especifica "ppop" como 01000 e executa LOADFUNCTION source OR dest -> dest

GNOP (- ¦ -) G
Especifica "ppop" como 01001 e executa LOADFUNCTION dest -> dest

GXOR (- ¦ -) G
Especifica "ppop" como 01010 e executa LOADFUNCTION source XOR dest -> dest

GNOTAND (- ¦ -) G
Especifica "ppop" como 01011 e executa LOADFUNCTION (NOT source) AND dest -> dest

WHITE (- ¦ -) G
Especifica "ppop" como 01100 e executa LOADFUNCTION "1" -> dest

GNOTOR (- ¦ -) G
Especifica "ppop" como 01101 e executa LOADFUNCTION (NOT source) AND dest -> dest

GNAND (- ¦ -) G
Especifica "ppop" como 01110 e executa LOADFUNCTION source NAND dest -> dest

GNOTS (- ¦ -) G
Especifica "ppop" como 01111 e executa LOADFUNCTION (NOT source) -> dest

GPLUS (- ¦ -) G
Especifica "ppop" como 10000 e executa LOADFUNCTION source + dest -> dest

GADDSAT (- ¦ -) G
Especifica "ppop" como 10001 e executa LOADFUNCTION ADDSAT(source,dest) -> dest

GMINUS (- | -) G
Especifica "ppop" como 10010 e executa LOADFUNCTION source - dest -> dest

GSUBSAT (- | -) G
Especifica "ppop" como 10011 e executa LOADFUNCTION SUBSAT(source,dest) -> dest

GMAX (- | -) G
Especifica "ppop" como 10100 e executa LOADFUNCTION MAX(source,dest) -> dest

GMIN (- | -) G
Especifica "ppop" como 10101 e executa LOADFUNCTION MIN(source,dest) -> dest

- Funções de ação gráfica:

COMPOSE (source dest | -) G
Compõe o stencil "source" com o "dest" , usando os definidores de contexto prévios. Ambos devem ser do mesmo tamanho. O stencil resultante é colocado em "dest".

PASTE (source dest | -) G
Compõe o stencil "source" em "dest" , usando os definidores de contexto prévios. O "source" é colocado na posição apontada pelo ponto corrente do stencil "dest". O stencil resultante está em "dest". O stencil "source" deve ser menor do que "dest" e após a operação não ultrapassar a fronteira desto. Após a operação , o ponto corrente de "dest" é o resultado da soma com o ponto corrente de "source".

DETACH (source dest | -) G
E' retirado do stencil source na posição apontada pelo seu ponto corrente, um stencil de dimensões iguais a "dest". A ação é efetuada usando os definidores de contexto prévios e o resultado é colocado no "dest".

REPASTE (source ny nx rdy rdx | -) G
Copia uma região dentro do stencil "source", de dimensões "rdy,rdx" , apontado pelo ponto corrente , na nova posição "ny,nx" , dentro do mesmo stencil . A ação gráfica é efetuada usando os definidores de contexto prévios. O ponto corrente passa a ser "ny,nx". As regiões não devem se sobrepor.

TRANSFER (source dest dy dx |m -) G
Copia uma região dentro do stencil "source" de dimensões "dy,dx" , apontada pelo ponto corrente de "source" , no stencil "dest", na posição apontada pelo ponto corrente de "dest". A ação é efetuada usando os definidores de contexto prévios.

LINEB (y x | -) G
Traça uma linha no stencil corrente , contido em CURSTENCIL , a partir do ponto corrente deste até a coordenada y,x (absoluta) , usando o stencil especificado em CURBRUSH como pincel . O ponto y,x passa a ser o novo ponto corrente. Rotina interna usado para a traçagem de trajetórias no DRAWMODE

LINEF (y x | -) G
Traça uma linha no stencil corrente , contido em CURSTENCIL , a partir do ponto corrente deste , até a coordenada y,x (absoluta) , usando o pixel de valor contido em BOUNDARY . O ponto y,x passa a ser o novo ponto corrente. Rotina interna usada para a traçagem de trajetórias no FILLMODE.

SFILL (TRUE yx yx .... | -) G
Rotina interna ,acessada por FILL , que executa o algoritmo de "scan line seed fill" , usado para pintar regiões delimitadas por LINEF. A rotina busca na pilha os pontos "sementes" até encontrar o valor TRUE. Os pontos yx são pares compactados em 32 bits , sendo o 16 msb correspondentes a coordenada y e o 16 lsb os correspondentes a coordenada x .

FILL (TRUE y x y x .... | -) G
Rotina preparatória para efetuar a pintura de regiões delimitadas por LINEF. Pega os pontos "sementes" até encontrar TRUE , compacta esses pares para o modo yx , prepara algumas variáveis internas e chama a rotina SFILL .

- Funções gráficas definidoras de formas:

RROTATION (y x | y x) G
Rotina de rotação de coordenadas . Pega os pontos y,x e rotaciona com relação a 0,0 o número de graus contidos em ROTATION. Põe na pilha os pontos transformados.

SCALE (y x | y x) G
Rotina de expansão de coordenadas . Pega os pontos y,x e efetua a transformação, conforme os parâmetros armazenados em SCALEX , SCALEY. As novas coordenadas são colocadas na pilha.

$x' = x.a/b$ a,b contidos em SCALEX
$y' = y.a/b$ a,b contidos em SCALEY

SHEAR (y x | y x) G

Rotina de deformação de coordenadas . Pega os pontos y,x e efetua a transformação, conforme os parâmetros armazenados em SHEARX , SHEARY. As novas coordenadas são colocadas na pilha.

$x' = x + y.a/b$ a,b contidos em SHEARX

$y' = y + x.a/b$ a,b contidos em SHEARY

AMOVETO (y x | -) G

Atualiza o ponto corrente do stencil corrente , na posição y,x em coordenadas absolutas do stencil

MOVETO (y x | -) G

Atualiza o ponto corrente do stencil corrente , na posição y,x ,em coordenadas relativas ao ponto ORIGIN , efetuando préviamente as transformações RROTATION, SCALE E SHEAR. Atualiza VCPT.

RMOVETO (y x | -) G

Atualiza o ponto corrente do stencil corrente , deslocando o ponto atual de y,x unidades , efetuando préviamente as transformações RROTATION , SCALE e SHEAR. Atualiza VCPT.

ALINETO (y x | -) G

Traça uma linha a partir do ponto corrente até o ponto y,x fornecido , no stencil corrente , em coordenadas absolutas. Se em DRAWMODE , executa LINEB . Se em FILLMODE , executa LINEF.

LINETO (y x | -) G

Traça uma linha a partir do ponto corrente até o ponto y,x, em coordenadas relativas ao ponto ORIGIN,efetuando préviamente as transformações RROTATION, SCALE e SHEAR. Atualiza VCPT . Se em DRAWMODE , executa LINEB , Se em FILLMODE , executa LINEF.

RLINETO (y x | -) G

Traça uma linha a partir do ponto corrente até o ponto deslocado de y,x unidades , efetuando préviamente as transformações RROTATION , SCALE e SHEAR. Atualiza VCPT. Se em DRAWMODE , executa LINEB , Se em FILLMODE , executa LINEF.

- Funções de conveniência para a impressão de Texto:

(string) G {"left-par"}
Especifica o texto que deve ser utilizado por BSHOW ou TSHOW. No modo interpretativo coloca o string seguinte, até o símbolo ")" , no "word buffer" e deixa na pilha o endereço e o comprimento. No modo compilativo , coloca o string no dicionário. Na execução coloca na pilha o endereço seguido do comprimento.

CURBMFONT (- ¦ adpf) G
Variável de controle que armazena o endereço "adf" da matriz codificadora do fonte de caracteres adotados para uso do comando BSHOW . O fonte deve ser constituído de stencils.

CURTFONT (- ¦ adpf) G
Variável de controle que armazena o endereço "adf" da matriz codificadora do fonte de caracteres adotados para uso do comando TSHOW . O fonte deve ser constituído de trajetórias.

ADJUSTXY (- ¦ adpf) G
Variável de controle dupla , que contém os incrementos y,x que serão usados para espacar cada caracter desenhado por BSHOW

BSHOW (addr n ¦ -) G
Pega o string do endereço "addr" , com "n" caracteres e usando CURBMFONT , escreve em CURSTENCIL , a partir de seu ponto corrente . Imprime o string usando fonte definido em stencils

SELECTBMFONT name G !
Coloca o "adf" da matriz "name" em CURTBMFONT.
Seleciona o fonte de caracteres em stencil , que será usado pelo comando BSHOW.

SELECTTFONT name G !
Coloca o "adf" da matriz "name" em CURTBMFONT.
Seleciona o fonte de caracteres em trajetórias , que será usado pelo comando TSHOW.

TSHOW (addr n ¦ -) G
Pega o string do endereço "addr" , com "n" caracteres e usando CURTFONT , escreve em CURSTENCIL a partir de seu ponto corrente . Imprime o string usando fonte definido em trajetórias .

- Funções auxiliares internas:

DOCREATE (- ¦ addr) R
R(addr ¦ -)
Transfere o endereço da sua célula de retorno para o topo da pilha de parâmetros . Rotina de execução das variáveis

DOCONSTANT (- ¦ x) R
R(addr ¦ -)
Transfere o conteúdo da sua célula de retorno para a pilha de parâmetros . Rotina de execução das constantes.

LIT (- ¦ x) R
Transfere o conteúdo da sua célula de retorno para a pilha de parâmetros e altera seu endereço de retorno para a célula seguinte a este . Rotina de execução para números compilados dentro de uma função no dicionário

BRANCH (- ¦ -) R
Transfere, incondicionalmente, a execução para o endereço contido na sua célula de retorno. Usado nas estruturas de controle.

?BRANCH (flag ¦ -) R
Se flag=FALSE , transfere a execução para o endereço contido na sua célula de retorno , caso contrário retorna para a célula seguinte. Usado nas estruturas de controle.

RDO (lmt indx ¦ -) R
!(- ¦ addr lmt indx)
Rotina inicializadora dos laços DO-LOOP. Põe na pilha de índices , o endereço de saída do loop , o índice final e o índice inicial.

RLOOP (- ¦ -) R
!(addr lmt indx ¦ addr lmt indx)
Rotina controladora dos laços DO-LOOP . Pega da pilha de índices o valor atual do índice , incrementa e compara com o limite. Caso seja menor , transfere a execução para o endereço contido na célula de retorno , caso contrário retorna na célula seguinte , eliminando os 3 números da pilha de índices.

RPLUSLOOP (x ¦ -) R
!(addr lmt indx ¦ addr lmt indx)
Idem a rotina anterior , porém com o incremento fornecido na pilha

RLEAVE (- ¦ -) R
!(addr !mt idx ¦ -)
Rotina de execução do comando LEAVE . Retorna a execução para o endereço contido na terceiro posição da pilha de índices , limpando-a.

RCLEAVE (flag ¦ -) R
!(addr !mt idx ¦ addr !mt idx)
Se flag=TRUE, executa RLEAVE , caso contrário retorna a execução para a célula de retorno

(FIND) (addr1 adh ¦ addr flag) R
Rotina de procura de uma palavra em um vocabulário.
Pega da pilha o adh da última palavra definida no vocabulário de interesse seguido do endereço do acumulador de palavras addr1 que contém a palavra em questão. A palavra em questão foi posicionada no acumulador pela função TOKEN.
A busca é feita até que a palavra seja encontrada ou que o adh de alguma definição contenha zero, ou seja fim de cadeia. A seguir, é colocado na pilha :
- adf da definição encontrada e flag=true, se a busca foi bem sucedida
- adf da definição encontrada e flag="1", se a busca foi bem sucedida e a palavra tem o atributo "imediato"
- endereço do acumulador de palavra e flag=false, se a busca não for bem sucedida.

(") (- ¦ addr n) R ("paren-quote")
retorna na pilha o endereço e o comprimento de um string colocado nas próximas células no dicionário. A execução continua na célula após o string.

(.") (- ¦ -) R ("paren-dot quote")
transfere o string que está colocado nas próximas células a seguir no dicionário , para o acumulador de saída. A execução continua na célula após o string

>MARK (- ¦ addr) R
reserva uma célula vazia no dicionário e põe o endereço desta célula na pilha . Usado por IF-THEN-ELSE, para marcar a posição de um "backward brach" futuro.

<MARK (- ¦ addr) R
põe o endereço da próxima célula vazia do dicionário na pilha . Usado por IF-THEN-ELSE , para marcar a posição de um "foward brach" futuro.

<RESOLVE (addr ¦ -) R
compila o endereço fornecido, na próxima célula do dicionário . Conclui um "backward brach" iniciado por >MARK

>RESOLVE (addr ¦ -) R
põe o endereço da próxima célula vazia do dicionário na célula de endereço fornecido . Conclui um "foward branch" iniciado por <MARK.

NUMBER (- ¦ x flag) R
converte o string no acumulador de palavra, no número binário correspondente , usando BASE . Se a transformação for bem sucedida , coloca o número seguido do flag=true. Caso contrário, coloca somente flag=false.

SPO (- ¦ addr) R
Constante do sistema que mantém o endereço inicial da pilha de parâmetros.

SP@ (- ¦ addr) R
Retorna na pilha o endereço atual do topo desta.

ABORT R DD
Inicializa pilha de parâmetros, e a pilha de retorno , e aciona INNER (também pode configurar atributos do hardware)

(ABORT) (flag ¦ -) R
Se flag=TRUE , imprime a mensagem que está nas células a seguir no dicionário e reinicializa o sistema através de ABORT.

STACK (- ¦ -) R
Testa se a pilha ultrapassou os limites estabelecidos, emitindo mensagem de erro correspondente e abortando o sistema. "Stack Underflow" , "Stack Overflow"

?MISSING (flag ¦ -) R
Pega o flag deixado da pilha por DEFINED , correspondente a busca da palavra que consta no "word buffer".
Se flag= FALSE é modo interpretativo, imprime o nome da palavra inválida, seguida de mensagem de erro e aborta.
Se flag= FALSE é modo compilativo , cancela a definição sendo montada no dicionário, imprime o nome da palavra inválida seguida de mensagem de erro e aborta Se flag=TRUE , nada efetua.

APENDICE 2) LISTAGEM DO PROGRAMA FONTE

A2.1) Introdução

E apresentado, a seguir, a listagem fonte da implementação aqui descrita do MINISCRIPT, no "Software Development Board" SDB340 da Texas Instruments.

Algumas funções tiveram seus nomes trocados , devido ao fato de o Assemblador não permitir certos caracteres. Porém , dentro dos cabeçalhos das funções , o nome original foi mantido.

Para a compreenção dos símbolos e abreviaturas usadas pelo assemblador , recomenda-se a leitura da Referência (Texas,87).

No fim da listagem, encontra-se a referência cruzada das definições.

Os nomes, na listagem, são sempre acrescidos de "H_", para indicar o cabeçalho. Assim, por exemplo, se o usuário deseja encontrar o cabeçalho de DUP, deve procurar H_DUP.

Alguns nomes foram escritos seguindo a sua "pronúncia". Por exemplo, para encontrar a função ."., deve-se procurar H_DOTQUOTE.

A função SHOWPAGE não consta na listagem, pois sua ação era a de simplesmente carregar o registrador interno DPYSTRT do TMS34010.

Sua implementação, na simulação apresentada no capítulo VII, foi :

```
H_SHOWPAGE:          .int elo no vocabulário gráfico
                     .byte 08h,0h
                     .string "SHOWPAGE"
                     .even
                     .field SHOWPAGE

SHOWPAGE:            .movi 0fffch,DPYSTRT
                     movi 13h, convsp
                     movi 13h, convdp
                     rets
```

Quando executada, aciona o display gráfico apresentando o conteúdo da memória gráfica, a partir do endereço 0.

MAS/rcps.

```
0001                    ******************************************************
0002                    *       MINISCRIPT                                   *
0003                    *       forth-pdl stefani 1989                       *
0004                    *       MARIO ANTONIO STEFANI - SC - 1990            *
0005                    *       versao 1.0 25/11/89                          *
0006                    *       revisao 2  14/01/90 nucleo 1                 *
0007                    *       revisao 3  21/01/90 nucleo 2                 *
0008                    *       revisao 4  28/01/90 entrada serial           *
0009                    *       revisao 5  19/05/90 modulo grafico           *
0010                    *        revisao 5.1 20/06/90                        *
0011                    *       revisao 5.2 28/06/90                         *
0012                    ******************************************************
0013                    *
0014                    *       constantes, simbolos
0015                    *
0019 00000000                   .data
0020          FFFFFFC0  eintl   .set    0ffffffc0h      ;vetor interrupcao
0021          C0000110  intenb  .set    0c0000110h      ;interrup enable reg.
0022          02000000  rrhr    .set    02000000h       ;read receiver holding reg. USART
0023          02001000  rsr     .set    02001000h       ;read status reg. USART
0024          02002000  rmr     .set    02002000h       ;read mode reg. USART
0025          02003000  rcr     .set    02003000h       ;read comand reg. USART
0026          02200000  wthr    .set    02200000h       ;write transmiter holding reg. USART
0027          02202000  wmr     .set    02202000h       ;write mode reg.USART
0028          02203000  wcr     .set    02203000h       ;write command reg. USART
0029              000B  pstk    .set    a11             ;parameter stack
0030              000C  astk    .set    a12             ;auxiliary stack
0031              000D  istk    .set    a13             ;index stack
0032              000F  rstk    .set    sp              ;return stack
0033          FFFFFFFF  true    .set    0ffffffffh      ;constante true
0034              0000  false   .set    0h              ;constante false
0035              5200  atksize         .set    5200h           ;tamanho das pilhas
0036              5200  i_pstk .set     atksize         ;tamanho pstk
0037              5200  i_astk .set     atksize         ;idem astk
0038              5200  i_istk .set     atksize         ;idem istk
0039              5200  i_rstk .set     atksize         ;idem rstk
0040 00000000                         .text
0041                    *
0042                    *       cold start
0043                    *
0044 00000000     C080  reset:          jauc rabort
     00000010 000093E0'
0045                    *
0046                    *       raiz do dicionario
0047                    *       inicio da descricao das funcoes
0048                    *       (veja descricao do uso no apendice)
0049                    *
0050                    *       uso dos registradores:  a11=parameter stack
0051                    *                               a12=auxiliar stack
0052                    *                               a13=index stack
0053                    *                               sp =return stack
0054                    *                               Com execao dos pointers de pilhas ,
0055                    *                               nenhum registrador e usado para a
0056                    *                               transferencia de dados entre funcoes.
0057                    *       Funcoes
0058                    *        convencao:     dados  funcao  resultados
0059                    *                       topo da pilha = elemento mais a direita
0060                    *                       f = flag
0061                    *                       - = nada
0062                    *
0063                    * x DUP x x     duplica topo da pilha
0064                    *
0065 00000030 00000000  H_DUP:        .int  0h                ;inicio do dicionario,
0066 00000050       03                .byte   03h, 0h         ;vocabulario CORE
     00000058       00
0067 00000060       44                .string  "DUP"
     00000068       55
     00000070       50
0068 00000080                         .even
0069 00000080 000000A0'               .field DUP
0070 000000A0     9560  DUP:          MOVE  *PSTK+,A0
0071 00000080     A00B                MOVE  A0,-*PSTK
0072 000000C0     A00B                MOVE  A0,-*PSTK
0073 000000D0     0960                RETS
0074                    *
0075                    * x DROP -     elimina topo da pilha
```

```
                                *
000000E0 00000030' H_DROP:       .int H_DUP
00000100        04               .byte    04h,0h
00000108        00
00000110        44               .string  "DROP"
00000118        52
00000120        4F
00000128        50
00000130                         .even
00000130 00000150'               .field drop
00000150      9560  DROP:        MOVE  *PSTK+, a0     ;(ADDK  32, PSTK)
00000160      0960               RETS
                                *
                                * x y SWAP y x intercambia os dois elementos do topo da pilha
                                *
00000170 000000E0' H_SWAP:       .int H_DROP
00000190        04               .byte   04h,0h
00000198        00
000001A0        53               .string "SWAP"
000001A8        57
000001B0        41
000001B8        50
000001C0                         .even
000001C0 000001E0'               .field swap
000001E0      9560  SWAP:        MOVE   *PSTK+,a0
000001F0      9561               MOVE   *PSTK+,A1
00000200      A00B               MOVE a0, -*PSTK
00000210      A02B               MOVE A1, -*PSTK
00000220      0960               RETS
                                *
                                * x y OVER x y x       copia no topo o segundo elemnto
                                *
00000230 00000170' H_OVER:       .int H_SWAP
00000250        04               .byte   04h,0h
00000258        00
00000260        4F               .string "OVER"
00000268        56
00000270        45
00000278        52
00000280                         .even
00000280 000002A0'               .field over
000002A0      B560  OVER:        MOVE    *PSTK(+32),A0      ;segundo elemento
000002B0      0020
000002C0      A00B               MOVE    a0,-*PSTK
000002D0      0960               RETS
                                *
                                * x y z ROT y z x      rotaciona terceiro elemento para topo
                                *
000002E0 00000230' H_ROT:          .int H_OVER
00000300        03                 .byte 03h,0h
00000308        00
00000310        52                 .string "ROT"
00000318        4F
00000320        54
00000330                           .even
00000330 00000350'                 .field rot
00000350      9560  ROT:           MOVE *PSTK+,a0
00000360      9561                 MOVE *PSTK+,A1
00000370      9562                 MOVE *PSTK+,A2
00000380      A02B                 MOVE A1,-*PSTK
00000390      A00B                 MOVE a0,-*PSTK
000003A0      A04B                 MOVE A2,-*PSTK
000003B0      0960                 RETS
                                *
                                * x NOT y       complemento de um
                                *
000003C0 000002E0' H_NOT:          .int H_ROT
000003E0        03                 .byte  03h,0h
000003E8        00
000003F0        4E                 .string "NOT"
000003F8        4F
00000400        54
00000410                           .even
00000410 00000430'                 .field not
00000430      9560  NOT:           MOVE  *PSTk+,a0
00000440      03E0                 NOT a0
```

```
00000450      A00B                    MOVE a0,-*PSTK
00000460      0960                    RETS
                          *
                          * x y AND z    and logico
                          *
00000470 000003C0' H_AND:             .int H_NOT
00000490        03                    .byte  03,0h
00000498        00
000004A0        41                    .string "AND"
000004A8        4E
000004B0        44
000004C0                              .even
000004C0 000004E0'                    .field and
000004E0      9560  AND:              MOVE *PSTK+,a0
000004F0      9561                    MOVE *PSTK+,A1
00000500      5020                    AND A1,A0
00000510      A00B                    MOVE A0,-*PSTK
00000520      0960                    RETS
                          *
                          * x y OR z     or logico
                          *
00000530 00000470' H_OR:              .int H_AND
00000550        02                    .byte 02h,0!
00000558        00
00000560        4F                    .string "OR"
00000568        52
00000570                              .even
00000570 00000590'                    .field or
00000590      9560  OR:               MOVE *PSTK+,a0
000005A0      9561                    MOVE *PSTK+,A1
000005B0      5420                    OR A1,A0
000005C0      A00B                    MOVE A0,-*PSTK
000005D0      0960                    RETS
                          *
                          * x y XOR z    exclusive or logico
                          *
000005E0 00000530' H_XOR:             .int H_OR
00000600        03                    .byte 03h,0h
00000608        00
00000610        58                    .string "XOR"
00000618        4F
00000620        52
00000630                              .even
00000630 00000650'                    .field xor
00000650      9560  XOR:              MOVE *PSTK+,a0
00000660      9561                    MOVE *PSTK+,A1
00000670      5620                    XOR A1,A0
00000680      A00B                    MOVE A0,-*PSTK
00000690      0960                    RETS
                          *
                          * x y < f      compara dois numeros , f=true se x<y
                          *
000006A0 000005E0' H_lt:               .int H_XOR
000006C0        01                    .byte 01h,0h
000006C8        00
000006D0        3C                    .string "<"
000006E0                              .even
000006E0 00000700'                    .field lt
00000700      5642  lt:               clr A2
00000710      9561                    MOVE *PSTK+,A1   ;(n2)
00000720      9560                    MOVE *PSTK+,A0   ;(n1)
00000730      4820                    CMP A1,A0
00000740      C501                    JRGE lt2         ; jumpe A0 GE A1 signed
00000750      1420                    subk 1,A0
00000760      A04B  lt2:              MOVE A2,-*PSTK
00000770      0960                    RETS
                          *
                          * x y = f      compara dois numeros , f=true se x=y
                          *
00000780 000006A0' H_eq:                 .int H_lt
000007A0        01                     .byte 01h,0h
000007A8        00
000007B0        3D                     .string "="
000007C0                               .even
000007C0 000007E0'                     .field eq
000007E0      5642  eq:                CLR A2
```

```
0199 000007F0      9561                    MOVE *PSTK+,A1
0200 00000800      9560                    MOVE *PSTK+,A0
0201 00000810      4801                    CMP A0,A1
0202 00000820      CB01                    JRNE eq2
0203 00000830      03E2                    NOT A2
0204 00000840      A04B   eq2:              MOVE A2, -*PSTK
0205 00000850      0960                    RETS
0206                      *
0207                      * x y > f        compara dois numeros , f=true se x>y
0208                      *
0209 00000860 00000780' H_gt:               .int H_eq
0210 00000880        01                    .byte 01h,0h
     00000888        00
0211 00000890        3E                    .string ">"
0212 000008A0                              .even
0213 000008A0 000008C0'                    .field gt
0214 000008C0      5642   gt:               CLR A2
0215 000008D0      9561                    MOVE *PSTK+,A1
0216 000008E0      9560                    MOVE *PSTK+,A0
0217 000008F0      4820                    Cmp A1,A0
0218 00000900      C601                    JRLE gt2
0219 00000910      03E2                    NOT A2
0220 00000920      A04B   gt2:              MOVE A2,-*PSTK
0221 00000930      0960                    RETS
0222                      *
0223                      * x 0< f        compara com zero , f=true se x<0
0224                      *
0225 00000940 00000860' H_lt0:              .int H_gt
0226 00000960        02                    .byte 02h,0h
     00000968        00
0227 00000970        30                    .string "0<"
     00000978        3C
0228 00000980                              .even
0229 00000980 000009A0'                    .field lt0
0230 000009A0      5642   lt0:              CLR A2
0231 000009B0      9560                    MOVE *PSTK+, A0
0232 000009C0      4840                    CMP A2,A0
0233 000009D0      C501                    JRGE lt02
0234 000009E0      03E2                    NOT A2
0235 000009F0      A04B   lt02:             MOVE A2,-*PSTK
0236 00000A00      0960                    RETS
0237                      *
0238                      * x 0= f        compara com zero , f=true se x=0
0239                      *
0240 00000A10 00000940' H_eq0:              .int H_lt0
0241 00000A30        02                    .byte 02,0h
     00000A38        00
0242 00000A40        30                    .string "0="
     00000A48        3D
0243 00000A50                              .even
0244 00000A50 00000A70'                    .field eq0
0245 00000A70      5642   eq0:              CLR A2
0246 00000A80      9560                    MOVE *PSTK+,A0
0247 00000A90      4840                    CMP A2,A0
0248 00000AA0      CB01                    JrNE eq02
0249 00000AB0      03E2                    NOT A2
0250 00000AC0      A04B   eq02:             MOVE A2, -*PSTK
0251 00000AD0      0960                    RETS
0252                      *
0253                      * x 0> f        compara com zero , f=true se x>0
0254                      *
0255 00000AE0 00000A10' H_gt0:              .int H_eq0
0256 00000B00        02                    .byte 02h,0h
     00000B08        00
0257 00000B10        30                    .string "0>"
     00000B18        3E
0258 00000B20                              .even
0259 00000B20 00000B40'                    .field gt0
0260 00000B40      5642   gt0:              CLR A2
0261 00000B50      9560                    MOVE *PSTK+,A0
0262 00000B60      4840                    CMP A2,A0
0263 00000B70      C601                    JRLE gt02
0264 00000B80      03E2                    NOT A2
0265 00000B90      A04B   gt02:             MOVE A2,-*PSTK
0266 00000BA0      0960                    RETS
0267                      *
```

```
0268                        * a @ x                 fetch , retorna o conteudo do endereco a
0269                        *
0270 00000BB0 00000AE0' H_fetch:      .int H_gt0
0271 00000BD0        01               .byte 01h,0h
     00000BD8        00
0272 00000BE0        40               .string  "@"
0273 00000BF0                         .even
0274 00000BF0 00000C10'               .field fetch
0275 00000C10      9561  fetch:       MOVE *PSTK+,A1
0276 00000C20      8420               MOVE *A1,A0
0277 00000C30      A00B               MOVE A0,-*PSTK
0278 00000C40      0960               RETS
0279                        *
0280                        * x a ! -      store , coloca em a o numero x
0281                        *
0282 00000C50 00000BB0' H_store:      .int H_fetch
0283 00000C70        01                .byte  01h,0h
     00000C78        00
0284 00000C80        21                .string "!"
0285 00000C90                          .even
0286 00000C90 00000CB0'                .field  store
0287 00000CB0      9561  store:        MOVE *PSTK+,A1
0288 00000CC0      9560                MOVE *PSTK+,A0
0289 00000CD0      8001                MOVE A0,*A1
0290 00000CE0      0960                RETS
0291                        *
0292                        * a C@ w       cfetch, retorna os 16 lsb contidos no endereco
0293                        *
0294 00000CF0 00000C50' H_Cfetch:     .int H_store
0295 00000D10        02               .byte 02h,0h
     00000D18        00
0296 00000D20        43               .string "C@"
     00000D28        40
0297 00000D30                         .even
0298 00000D30 00000D50'               .field Cfetch
0299 00000D50      9561  Cfetch:      MOVE *PSTK+,A1
0300 00000D60      8420               MOVE *A1,A0
0301 00000D70      0B80               ANDI  0FFFFh,a0
     00000D80 FFFF0000
0302 00000DA0      A00B               MOVE A0,-*PSTK
0303 00000DB0      0960               RETS
0304                        *
0305                        * x a C! -      cstore, armazena os 16 lsb de x em a
0306                        *
0307 00000DC0 00000C50' H_Cstore:     .int H_store
0308 00000DE0        02               .byte 02h,0h
     00000DE8        00
0309 00000DF0        43               .string "C!"
     00000DF8        21
0310 00000E00                         .even
0311 00000E00 00000E20'               .field Cstore
0312 00000E20      9561  Cstore:      MOVE *PSTK+,A1
0313 00000E30      9560               MOVE *PSTK+,A0
0314 00000E40      018E               GETST A14
0315 00000E50      0550               SeTF 16,0,0
0316 00000E60      8001               MOVE A0,*A1
0317 00000E70      01AE               PUTST A14
0318 00000E80      0960               RETS
0319                        *
0320                        * x a +! -      acumula , soma x ao conteudo de a
0321                        *
0322 00000E90 00000DC0' H_plustore:   .int H_Cstore
0323 00000EB0        02               .byte 02h,0h
     00000EB8        00
0324 00000EC0        2B               .string "+!"
     00000EC8        21
0325 00000ED0                         .even
0326 00000ED0 00000EF0'               .field plustore
0327 00000EF0      9562  plustore:    MOVE *PSTK+,A2     ; addr
0328 00000F00      9561               MOVE *PSTK+,A1     ; w
0329 00000F10      8440               MOVE *A2,a0
0330 00000F20      4020               ADD  A1,a0
0331 00000F30      8002               MOVE a0,*A2
0332 00000F40      0960               RETS
0333                        *
0334                        * a EXECUTE     executa a rotina do endereco a
```

```
                                *
00000F50 00000E90' H_EXECUTE:      .int H_plustore
00000F70        07                 .byte 07h,0h
00000F78        00
00000F80        45                 .string "EXECUTE"
00000F88        58
00000F90        45
00000F98        43
00000FA0        55
00000FA8        54
00000FB0        45
00000FC0                           .even
00000FC0 00000FE0'                 .field EXECUTE
00000FE0      9560  EXECUTE:       MOVE *PSTK+,A0
00000FF0      0160                 jump A0            ; (call A0)
00001000      0960                 RETS                ;      jump mais rapido
                                *
                                * - DOCREATE a Rotina auxiliar para VARIABLE , poe endereco 'a'
                                *                da proxima celula de memoria na pilha
                                *
00001010 00000F50' H_DOCREATE:  .int  H_execute
00001030        08                .byte 08h,0h
00001038        00
00001040        44                .string "DOCREATE"
00001048        4F
00001050        43
00001058        52
00001060        45
00001068        41
00001070        54
00001078        45
00001080                          .even
00001080 000010A0'                .field DOCREATE
000010A0      95E0  DOCREATE:     MOVE *SP+,a0
000010B0      A00B                MOVE a0,-*PSTK
000010C0      0960                RETS
                                *
                                * - DOCONSTANT n        Rotina auxiliar para CONSTANT , poe
                                *                       conteudo da proxima celula de memoria
                                *                       na pilha
                                *
000010D0 00001010' H_DOCONSTANT:  .int H_DOCREATE
000010F0        0A                .byte  10,0
000010F8        00
00001100        44                .string "DOCONSTANT"
00001108        4F
00001110        43
00001118        4F
00001120        4E
00001128        53
00001130        54
00001138        41
00001140        4E
00001146        54
00001150                          .even
00001150 00001170'                .field DOCONSTANT
00001170      95E1  DOCONSTANT:   MOVE *SP+,A1
00001180      8420                MOVE *A1,a0
00001190      A00B                MOVE a0,-*PSTK
000011A0      0960                RETS
                                *
                                * - (LIT) n     rotina auxiliar, recupera numeros compilados.
                                *                n e' o conteudo da proxima celula no dicionario
                                *
000011B0 000010D0' H_lit:       .int H_DOCONSTANT
000011D0        05                .byte 05h,0h
000011D8        00
000011E0        28                .string "(LIT)"
000011E8        4C
000011F0        49
000011F8        54
00001200        29
00001210                          .even
00001210 00001230'                .field lit
00001230      85E1  lit:          move *sp,a1        ;sp aponta para o numero
00001240      8420                MOVE *A1,a0
```

```
00001250      1001                ADDk 32,a1
00001260      802F               move a1,*SP      ; passa sobre dado
00001270      A00B                MOVE a0,-*PSTK
00001280      0960                RETS
                        *
                        * - SPC 20h    coloca na pilha o ascii de space
                        *
00001290 000011B0' H_spc:           .int H_lit
000012B0        03                  .byte 03,0h
000012B8        00
000012C0        53                  .string "SPC"
000012C8        50
000012D0        43
000012E0                            .even
000012E0 00001300'                  .field  spc
00001300      09C0  spc:            movi 20h,a0
00001310      0020
00001320      A00B                  move a0,-*pstk
00001330      0960                  rets
                        *
                        * - TRUE f     coloca na pilha o numero booleano true
                        *
00001340 00001290' H_tru:           .int H_spc
00001360        04                  .byte 04,0h
00001368        00
00001370        54                  .string "TRUE"
00001378        52
00001380        55
00001388        45
00001390                            .even
00001390 000013B0'                  .field tru
000013B0      09C0  tru:            movi -1,a0
000013C0      FFFF
000013D0      A00B                  move a0,-*pstk
000013E0      0960                  rets
                        *
                        * - FALSE f    coloca na pilha o numero booleano false
                        *
000013F0 00001340' H_fal:           .int H_tru
00001410        05                  .byte 05,0h
00001418        00
00001420        46                  .string "FALSE"
00001428        41
00001430        4C
00001438        53
00001440        45
00001450                            .even
00001450 00001470'                  .field fal
00001470      09C0  fal:            movi 0,a0
00001480      0000
00001490      A00B                  move a0,-*pstk
000014A0      0960                  rets
                        *
                        * - BRANCH -   rotina auxiliar usado por DO , IF
                        *             salta execucao para o endereco contido na proxima
                        *             celula no dicionario
                        *
000014B0 000013F0' H_BRANCH:        .int H_fal
000014D0        06                  .byte 06,0h
000014D8        00
000014E0        42                  .string "BRANCH"
000014E8        52
000014F0        41
000014F8        4E
00001500        43
00001508        48
00001510                            .even
00001510 00001530'                  .field BRANCH
00001530      85E1  BRANCH:         MOVE *SP,A1
00001540      8420                  MOVE *A1,a0
00001550      800F                  MOVE a0,*SP
00001560      0960                  RETS
                        *
                        * f ?BRANCH -   rotina auxiliar usada por DO , IF
                        *             salta execucao para o endereco contido na proxima
                        *             celula no dicionario se f for false
```

```
0436                          *
0437 00001570 000014B0' H_cbranch:      .int H_BRANCH
0438 00001590        07                 .byte 07h,0h
     00001598        00
0439 000015A0        3F                 .string  "?BRANCH"
     000015A8        42
     000015B0        52
     000015B8        41
     000015C0        4E
     000015C8        43
     000015D0        48
0440 000015E0                           .even
0441 000015E0 00001600'                 .field cbranch
0442 00001600      85E1  cbranch:       MOVE *SP,A1
0443 00001610      4C22                 MOVE A1,A2
0444 00001620      8420                 MOVE *A1,A0
0445 00001630      800F                 MOVE A0,*SP
0446 00001640      9560                 MOVE *PSTK+,A0
0447 00001650      0B40                 CMPI 0,A0
     00001660      FFFF
0448 00001670      CA02                 JRZ cbranchSAI
0449 00001680      1002                 ADDK 32,A2
0450 00001690      804F                 MOVE A2,*SP
0451 000016A0      0960  cbranchSAI:   RETS
0452                          *
0453                          * - HERE a      retorna o endereco da proxima celula livre
0454                          *              no dicionario
0455                          *
0456 000016B0 00001570' H_HERE:         .int H_cbranch
0457 000016D0        04                 .byte 04h,0h
     000016D8        00
0458 000016E0        48                 .string "HERE"
     000016E8        45
     000016F0        52
     000016F8        45
0459 00001700                           .even
0460 00001700 00001720'                 .field HERE
0461 00001720      05A0  HERE:          MOVE @fDP,A0
     00001730 00001810'
0462 00001750      A00B                 MOVE A0,-*PSTK
0463 00001760      0960                 RETS
0464                          *
0465                          * - DIP a       variavel de controle , contem em a o pointer
0466                          *              do dicionario
0467                          *
0468 00001770 000016B0' H_DIP:          .int H_HERE
0469 00001790        03                .byte 03,0h
     00001798        00
0470 000017A0        44                .string "DIP"
     000017A8        49
     000017B0        50
0471 000017C0                          .even
0472 000017C0 000017E0'                .field DIP
0473 000017E0      0D5F  DIP:          calla docreate
     000017F0 000010A0'
0474 00001810 00015430' fDP:           .field endp ;(cria espaco)
0475                          *
0476                          * n ALLOT -     reserva n celulas no dicionario , movimentando DIP
0477                          *
0478 00001830 00001770' H_ALLOT:        .int H_DIP
0479 00001850        05                 .byte 05h,0h
     00001858        00
0480 00001860        41                 .string "ALLOT"
     00001868        4C
     00001870        4C
     00001878        4F
     00001880        54
0481 00001890                           .even
0482 00001890 000018B0'                 .field ALLOT
0483 000018B0      9560  ALLOT:         MOVE *PSTK+,a0
0484 000018C0      2080                 SLA 4,a0
0485 000018D0      05A1                 MOVE @fdp,A1
     000018E0 00001810'
0486 00001900      4001                 ADD a0,A1
0487 00001910      0581                 MOVE A1, @fdp
     00001920 00001810'
```

```
00001940      0960                    RETS
                        *
                        * x , -                  comma , coloca o valor n na proxima celula do dic.
                        *
00001950 00001830' H_comma:            .int H_ALLOT
00001970        01                   .byte 01h,0h
00001978        00
00001980        2C                   .string ","
00001990                             .even
00001990 000019B0'                   .field comma
000019B0      9560  comma:           MOVE *PSTK+,a0
000019C0      05A1                   MOVE @fdp,A1
000019D0 00001810'
000019F0      9001                   MOVE a0,*A1+
00001A00      0581                   MOVE A1, @fdp
00001A10 00001810'
00001A30      0960                   RETS
                        *
                        * n C, -       c-comma, armazena os 16 lsb de n na proxima celula do
                        *               dicionario
                        *
00001A40 00001950' H_ccomma:           .int H_comma
00001A60        02                   .byte 02h,0h
00001A68        00
00001A70        43                   .string "C,"
00001A78        2C
00001A80                             .even
00001A80 00001AA0'                   .field ccomma
00001AA0      05A1  ccomma:          MOVE @fdp,A1
00001AB0 00001810'
00001AD0      9560                   MOVE *PSTK+,a0
00001AE0      018E                  getst a14
00001AF0      0550                  setf 16,0,0
00001B00      8001                   MOVE a0,*A1
00001B10      1201                   ADDk 16,A1
00001B20      01AE                  putst a14
00001B30      0581                   MOVE A1,@fdp
00001B40 00001810'
00001B60      0960                   RETS
                        *
                        * - COMPILE fun                 compila no dic. a proxima funcao fun presente
                        *                        na definicao sendo executada
                        *
00001B70 00001A40' H_COMPile:          .int H_ccomma
00001B90        07                     .byte 07h,0h
00001B98        00
00001BA0        43                     .string "COMPILE"
00001BA8        4F
00001BB0        4D
00001BB8        50
00001BC0        49
00001BC8        4C
00001BD0        45
00001BE0                               .even
00001BE0 00001C00'                     .field COMPILE
00001C00      85E1  COMPILE:           MOVE *SP,A1
00001C10      05A2                     MOVF @fdp,A2
00001C20 00001810'
00001C40      018E                     gETST A14
00001C50      0550                     SETf 16,0,0
00001C60      9822                     MOVE *A1+,*A2+
00001C70      01AE                     PUTST A14
00001C80      9822                     MOVE *A1+,*A2+
00001C90      0582                     MOVE A2,@fdp
00001CA0 00001810'
00001CC0      802F                     MOVE A1,*SP
00001CD0      0960                     RETS
                        *
                        * 1 i (DO) -    rotina auxiliar de controle dos lacos DO,LOOP
                        *               coloca na pilha auxiliar o endereco de saida
                        *
00001CE0 00001B70' H_rdo:            .int H_COMPILE
00001D00        04                   .byte D4h,0h
00001D08        00
00001D10        28                   .string "(DO)"
00001D18        44
```

```
      00001D20      4F
      00001D28      29
 0546 00001D30                          .even
 0547 00001D30 00001D50'                .field rdo
 0548 00001D50      9560  rdo:          MOVE *PSTK+,a0      ; indice
 0549 00001D60      9561                MOVE *PSTK+,A1      ; limit
 0550 00001D70      85E2                MOVE *SP,A2
 0551 00001D80      8443                MOVE *A2,A3         ; endereco depois do (DO)
 0552 00001D90      1D02                addk 32,a2
 0553 00001DA0      804F               move a2,*sp         ; contem o endereco de jump
 0554                     *                                 ; endereco de retorno
 0555 00001DB0      A06C                MOVE A3,-*ASTK      ; salva na pilha auxiliar o endereco
 0556                     *                                 ; de saida
 0557 00001DC0      A02C                MOVE A1,-*ASTK     ; limite
 0558 00001DD0      A00C                MOVE a0,-*ASTK      ; indice
 0559 00001DE0      0960                RETS
 0560                     *
 0561                     * - (LOOP) -   rotina auxiliar de controle dos lacos DO,LOOP
 0562                     *              Pula para a palavra imediatamente apos (DO) se
 0563                     *              o indice nao ultrapassou o limite .
 0564                     *
 0565 00001DF0 00001CED' H_rloop:      .int H_rdo
 0566 00001E10      06                 .byte 06h,0h
      00001E18      00
 0567 00001E20      28                 .string "(LOOP)"
      00001E28      4C
      00001E30      4F
      00001E38      4F
      00001E40      50
      00001E48      29
 0568 00001E50                          .even
 0569 00001E50 00001E70'                .field rloop
 0570 00001E70      09C3  rloop:       MOVi 1,A3
      00001E80      0001
 0571 00001E90      9580  rloop1:      MOVE *ASTK+,A0       ;index
 0572 00001EA0      9581                MOVE *ASTK+,A1       ;limit
 0573 00001EB0      4060                ADD A3,a0
 0574 00001EC0      4820                CMP a1,A0
 0575 00001ED0      C504                JRGE rloop2
 0576 00001EE0      A02C                MOVE A1,-*ASTK
 0577 00001EF0      A00C                MOVE a0,-*ASTK
 0578 00001F00      C000                JRuC branch
      00001F10      FF61
 0579 00001F20      9583  rloop2:      MOVE *ASTK+,A3
 0580 00001F30      85E0                MOVE *SP,a0
 0581 00001F40      1000                addk 32,a0
 0582 00001F50      800F                MOVE a0,*SP
 0583 00001F60      0960                RETS
 0584                     *
 0585                     * x (+LOOP) -  rotina auiliar usada por +LOOP , similar a LOOP porem
 0586                     *              com incremento x fornecido
 0587 00001F70 00001DF0' H_rplusloop:    .int H_rloop
 0588 00001F90      07                 .byte 07,0h
      00001F98      00
 0589 00001FA0      28                 .string "(+LOOP)"
      00001FA8      2B
      00001FB0      4C
      00001FB8      4F
      00001FC0      4F
      00001FC8      50
      00001FD0      29
 0590 00001FE0                          .even
 0591 00001FE0 00002000'                .field rplusloop
 0592 00002000      9563  rplusloop:   MOVE *PSTK+,A3
 0593 00002010      C0E7               JRUC rloop1
 0594                     *
 0595                     * - I i                 retorna o indice corrente do laco DO-LOOP mais
 0596                     *              interno
 0597                     *
 0598 00002020 00001F70' H_I:           .int H_rplusloop
 0599 00002040      01                 .byte 01,0h
      00002048      00
 0600 00002050      49                 .string "I"
 0601 00002060                          .even
 0602 00002060 00002080'                .field I
 0603 00002080      8580  I:           MOVE *ASTK,a0
```

```
00002090      A00B                MOVE a0,-*PSTK
000020A0      0960                RETS
                        *
                        * - J j                 retorna o indice do segundo laco DO-LOOP mais interno
                        *
                        *
000020B0 00002020' H_J:           .int H_I
000020D0        01                .byte 01h,0h
000020D8        00
000020E0        4A                .string "J"
000020F0                          .even
000020F0 00002110'                .field J
00002110      B580  J:            MOVE *aSTK(+32),a0
00002120      0020
00002130      A00B                MOVE a0,-*PSTK
00002140      0960                RETS
                        *
                        * - K k                 idem para o terceiro laco
                        *
00002150 000020B0' H_K:           .int H_J
00002170        01                .byte 01h,0h
00002178        00
00002180        4B                .string "K"
00002190                          .even
00002190 000021B0'                .field K
000021B0      B580  K:            MOVE *ASTK(+64),a0
000021C0      0040
000021D0      A00B                MOVE a0,-*PSTK
000021E0      0960                RETS
                        *
                        * - (LEAVE)    abandona incondicionalmente um laco DO-LOOP
                        *
000021F0 00002150' H_rleave:      .int H_K
00002210        07                .byte 07h,0h
00002218        00
00002220        28                .string "(LEAVE)"
00002228        4C
00002230        45
00002238        41
00002240        56
00002248        45
00002250        29
00002260                          .even
00002260 00002280'                .field rleave
00002280      0B0C  rleave:       addi 64,ASTK  ; index, limit   popped
00002290      0040
000022A0      9580                MOVE *ASTK+,a0 ; endereco de retorno
000022B0      800F                MOVE a0,*SP
000022C0      0960                RETS
                        *
                        * f (?LEAVE) - abandona um laco DO-LOOP se f = false
                        *
000022D0 000021F0' H_rcleave:     .int H_rleave
000022F0        08                .byte 08h,0h
000022F8        00
00002300        28                .string "(?LEAVE)"
00002308        3F
00002310        4C
00002318        45
00002320        41
00002328        56
00002330        45
00002338        29
00002340                          .even
00002340 00002360'                .field rcleave
00002360      9560  rcleave:      MOVE *PSTK+,a0
00002370      5400                OR   a0,a0
00002380      CBEF                JRNZ rleave
00002390      0960                RETS
                        *
                        * a TOKEN -    transmite ao acumulador de saida o string iniciado
                        *             no endereco a .
                        *
000023A0 000022D0' h_type:                .int H_rcleave
000023C0        04                .byte 04h,0h
000023C8        00
```

```
0659 000023D0          74                    .string "type"
     000023D8          79
     000023E0          70
     000023E8          65
0660 000023F0                                .even
0661 000023F0 00002410'                      .field type
0662 00002410       9560  type:              move *pstk+,a0
0663 00002420       8E01                     movb *a0,a1
0664 00002430       1200                     addk 16,a0
0665 00002440       0B81                     andi 0ffh,a1
     00002450 FFFFFF00
0666 00002470       0B41                     cmpi 0,a1
     00002480       FFFF
0667 00002490       CA0A                     jrz typesai
0668 000024A0       05A2                     move @ftobp,a2
     000024B0 000074B0'
0669 000024D0       9C02  typemove:          movb *a0,*a2
0670 000024E0       1100                     addk 8,a0
0671 000024F0       1102                     addk 8,a2
0672 00002500       3C81                     dsj a1,typemove
0673 00002510       0582                     move a2,@ftobp
     00002520 000074B0'
0674 00002540       0960  typesai:           rets
0675                      *
0676                      * - CONTEXT a     retorna o endereco da variavel que contem o vocabulario
0677                      *                  de contexto
0678                      *
0679 00002550 000023A0' H_CONTEXT:           .int H_TYPE
0680 00002570         07                     .byte 07h,0h
     00002578         00
0681 00002580         43                     .string "CONTEXT"
     00002588         4F
     00002590         4E
     00002598         54
     000025A0         45
     000025A8         58
     000025B0         54
0682 000025C0                                .even
0683 000025C0 000025E0'                      .field CONTEXT
0684 000025E0       0D5F  CONTEXT:           calla docreate
     000025F0 000010A0'
0685 00002610 00002B10' fCONTEXT:            .field fCORE
0686                      *
0687                      * - CURRENT a      idem para vocabulario corrente
0688                      *
0689 00002630 00002550' H_CURRENT:           .int H_CONTEXT
0690 00002650         07                     .byte 07h,0h
     00002658         00
0691 00002660         43                     .string "CURRENT"
     00002668         55
     00002670         52
     00002678         52
     00002680         45
     00002688         4E
     00002690         54
0692 000026A0                                .even
0693 000026A0 000026C0'                      .field CURRENT
0694 000026C0       0D5F  CURRENT:           calla docreate
     000026D0 000010A0'
0695 000026F0 00002B10' fcurrent:            .field fCORE
0696                      *
0697                      * - ENTRY a       retorna o endereco do cabelhaco da ultima definicao
0698                      *                  no vocabulario corrente
0699                      *
0700 00002710 00002630' H_ENTRY:             .int H_CURRENT
0701 00002730         05                     .byte 05h,0h
     00002738         00
0702 00002740         45                     .string "ENTRY"
     00002748         4E
     00002750         54
     00002758         52
     00002760         59
0703 00002770                                .even
0704 00002770 00002790'                      .field ENTRY
0705 00002790       0D5F  ENTRY:             calla current     ;current address
     000027A0 000026C0'
```

```
000027C0      0D5F                 calla fetch            ;vocabulary address
000027D0 00000C10'
000027F0      0D5F                 calla fetch            ;header address
00002800 00000C10'
00002820      0960                 RETS
                        *
                        * x y + z      soma
                        *
00002830 00002710' H_plus:           .int H_ENTRY
00002850        01                 .byte 01h,0h
00002858        00
00002860        2B                 .string "+"
00002870                           .even
00002870 00002890'                 .field plus
00002890      9560  plus:          MOVE *PSTK+,a0
000028A0      9561                 MOVE *PSTK+,A1
000028B0      4001                 ADD a0,A1
000028C0      A02B                 MOVE A1,-*PSTK
000028D0      0960                 RETS
                        *
                        * x y - z      subtracao
                        *
000028E0 00002830' H_minus:           .int H_plus
00002900        01                 .byte 01h,0h
00002908        00
00002910        2D                 .string "-"
00002920                           .even
00002920 00002940'                 .field minus
00002940      9560  minus:         MOVE *PSTK+,a0
00002950      9561                 MOVE *PSTK+,A1
00002960      4401                 SUB a0,A1
00002970      A02B                 MOVE A1,-*PSTK
00002980      0960                 RETS
                        *
                        * - COMPILER a retorna o endereco da variavel que contem o vocabulario
                        *                do compilador
                        *
00002990 000028E0' H_COMPILER:       .int H_minus
000029B0        08                   .byte 08h,0h
000029B8        00
000029C0        43                   .string "COMPILER"
000029C8        4F
000029D0        4D
000029D8        50
000029E0        49
000029E8        4C
000029F0        45
000029F8        52
00002A00                             .even
00002A00 00002A20'                   .field COMPILER
00002A20      0D5F  COMPILER:        calla docreate
00002A30 000010A0'
00002A50 0000AC00' fCOMPILER:        .field H_bracktick     ;topo do voc compiler
                        *
                        * - CORE a      idem para o vocabulario raiz
                        *
00002A70 00002990' H_CORE:         .int H_COMPILER
00002A90        04                 .byte 04h,0h
00002A98        00
00002AA0        43                 .string "CORE"
00002AA8        4F
00002AB0        52
00002AB8        45
00002AC0                           .even
00002AC0 00002AE0'                 .field CORE
00002AE0      0D5F  CORE:          calla docreate
00002AF0 000010A0'
00002B10 0000B970' fCORE:          .field H_graphics          ;topo do voc core
                        *
                        * - LAST a      retorna o endereco da variavel que contem
                        *                ultima definicao no dicionario
                        *
00002B30 00002A70' H_LAST:         .int H_CORE
00002B50        04                 .byte 04,0h
00002B58        00
00002B60        4C                 .string "LAST"
```

```
      00002B68        41
      00002B70        53
      00002B78        54
 0763 00002B80                          .even
 0764 00002B80 00002BA0'                field LAST
 0765 00002BA0      0D5F  LAST:         calla docreate
      00002BB0 000010A0'
 0766 00002BD0 00014D20' fLAST:         .field H_tshow    ;ultima palavra no dic
 0767                      *
 0768                      * - COMPILERWORD -
 0769                      *                 poe a ultima definicao no vocabulario COMPILER
 0770                      *
 0771 00002BF0 00002A70' H_COMPILERWORD:    .int H_CORE
 0772 00002C10        09                      .byte 09h,0h
      00002C18        00
 0773 00002C20        43                      .string "COMPILERWORD"
      00002C28        4F
      00002C30        4D
      00002C38        50
      00002C40        49
      00002C48        4C
      00002C50        45
      00002C58        52
      00002C60        57
      00002C68        4F
      00002C70        52
      00002C78        44
 0774 00002C80                              .even
 0775 00002C80 00002CA0'                    .field COMPILERWORD
 0776 00002CA0      0D5F  COMPILERWORD:     calla entry      ;latest header address
      00002CB0 00002790'
 0777 00002CD0      0D5F                    calla dup
      00002CE0 000000A0'
 0778 00002D00      0D5F                    calla dup
      00002D10 000000A0'
 0779 00002D30      0D5F                    calla fetch          ;penultimo link
      00002D40 00000C10'
 0780 00002D60      0D5F                    calla current
      00002D70 000026C0'
 0781 00002D90      0D5F                    calla fetch
      00002DA0 00000C10'
 0782 00002DC0      0D5F                    calla store
      00002DD0 00000CB0'
 0783 00002DF0      0D5F                    calla compiler    ;/ core
      00002E00 00002A20'
 0784 00002E20      0D5F                    calla fetch
      00002E30 00000C10'
 0785 00002E50      0D5F                    calla swap
      00002E60 000001E0'
 0786 00002E80      0D5F                    calla store
      00002E90 00000CB0'
 0787 00002EB0      0D5F                    calla compiler    ;/ core
      00002EC0 00002A20'
 0788 00002EE0      0D5F                    calla store
      00002EF0 00000CB0'
 0789 00002F10      0960                    RETS
 0790                      *
 0791                      * c TOKEN -     retira de TIB a proxima palavra delimitada pelo
 0792                      *                caracter c , colocando no word-buffer.
 0793                      *
 0794 00002F20 00002BF0' H_token:       .int h_COMPILERWORD
 0795 00002F40        05                .byte 05h,0h
      00002F48        00
 0796 00002F50        54                .string "TOKEN"
      00002F58        4F
      00002F60        4B
      00002F68        45
      00002F70        4E
 0797 00002F80                          .even
 0798 00002F80 00002FA0'                .field TOKEN
 0799 00002FA0      05AA  TOKEN:         MOVE @fdp, A10    ;DP aponta para o link field
      00002FB0 00001810'
 0800 00002FD0      100A                 ADDK 32,A10       ;word buffer
 0801 00002FE0      4D49                 MOVE A10,a9      ;a09 aponta para Nchar field
 0802 00002FF0      120A                 ADDK 16,A10       ;A10 aponta para nome field
 0803 00003000      05A8                 MOVE @finp,a8    ;a08 aponta p/ o pointer do imput buffer
```

```
      00003010 00007630'
0804  00003030     9565                MOVE *PSTK+,A5  ;pega separador adotado
0805  00003040     0B85                ANDI 0FFh,A5   ;limpo
      00003050 FFFFFF00
0806                      *             MOVI 20h,A4     ;A4 contem SPC
0807                      *             CMP A4,A5       ;o separador e SPC
0808                      *             JRNE TOK2       ;se nao token start
0809  00003070     8F03   TOK1:        MOVb *A8,A3     ;compara se e separador
0810  00003080     0B83                ANDI 0FFh,A3    ;compara se fim buffer
      00003090 FFFFFF00
0811  000030B0     0B43               cmpi 00,a3
      000030C0     FFFF
0812  000030D0     CA24                jrz tokenbuf    ;fim buf
0813  000030E0     4865                CMP A3,A5
0814  000030F0     CB02                JRNE TOK2       ;se nao token start
0815  00003100     1108                ADDK 08,A8      ;incremente a pointer
0816  00003110     C0F5                JRUC TOK1
0817  00003120     4D07   TOK2:        MOVE A8,A7      ;salva taken start
0818  00003130     4400                SUB a0,a0       ;zero contador
0819  00003140     1020   TOK3:        ADDK 01,a0
0820  00003150     1108                ADDK 08,A8
0821  00003160     8F03                MOVb *A8,A3
0822  00003170     4865                CMP A3,A5       ;e = separador
0823  00003180     CA05                JREQ TOK4      ;se for vai para o fim
0824  00003190     0B43                cmpi 00,a3
      000031A0     FFFF
0825  000031B0     CBF8                jrne tok3
0826                      *             movi true,a6
0827  000031C0     0BE8                SUBI 08,A8
      000031D0     FFF7
0828  000031E0     0B08   TOK4:        ADDI 08,A8      ;passa pelo separador
      000031F0     0008
0829  00003200     0588                MOVE A8,@finp    ;atualiza imput buffer pointer
      00003210 00007630'
0830  00003230     4C01                MOVE a0,A1      ;copia contagem
0831  00003240     9CEA   TOK5:        MOVb *A7,*A10   ;imput buffer - nome field
0832  00003250     1107                ADDK 8,A7       ;incrementa pointers
0833  00003260     110A                ADDK 8,A10
0834  00003270     3C81                DSJ A1,TOK5
0835  00003280     1FE0                BTST 0,A0       ;testa se a contagem e par
0836  00003290     CA02                JRZ TOK6        ;se impar incrementa contagem de 1
0837                      *             ADDk 1,a0       ;nao arredonda?
0838  000032A0     8CAA                MOVb A5,*a10     ;poe separador no nome field
0839  000032B0     110A                ADDK 8,A10
0840  000032C0     8C09   TOK6:        MOVb a0,*A9     ;poe contagem no Nchar field
0841  000032D0     1109                addk 8,a9
0842  000032E0     09C6                movi 0,a6
      000032F0     0000
0843  00003300     8CC9                movb a6,*a9
0844  00003310     0960                RETS
0845  00003320     09C6   tokenbuf:    movi false,a6
      00003330     0000
0846  00003340     0586                move a6,@fnendbuf
      00003350 00009330'
0847  00003370     0960                rets
0848                      *
0849                      * - CREATE -  cria no dicionario um cabecalho com o nome
0850                      *            que vem a seguir em TIB
0851                      *
0852  00003380 00002F20' H_CREATE:     .int H_TOKEN
0853  000033A0       05                .byte 05,0h
      000033A8       00
0854  000033B0       43                .string "CREATE"
      000033B8       52
      000033C0       45
      000033C8       41
      000033D0       54
      000033D8       45
0855  000033E0                         .even
0856  000033E0 00003400'               .field CREATE
0857  00003400     0D5F   CREATE:      calla spc
      00003410 00001300'
0858  00003430     0D5F                calla token
      00003440 00002FA0'
0859  00003460     0D5F                calla entry
      00003470 00002790'
```

```
0860 00003490      0D5F                    calla here
     000034A0 00001720'
0861 000034C0      0D5F                    calla dup
     000034D0 000000A0'
0862 000034F0      0D5F                    calla last
     00003500 00002BA0'
0863 00003520      0D5F                    calla store
     00003530 00000CB0'
0864 00003550      0D5F                    calla current
     00003560 000026C0'
0865 00003580      0D5F                    calla fetch         ;endereco do vocabulario current
     00003590 00000C10'
0866 000035B0      0D5F                    calla store         ;salva no vocabulario
     000035C0 00000CB0'
0867 000035E0      0D5F                    calla comma         ;salva no dicionario ultimo link
     000035F0 000019B0'
0868 00003610      05A0                    MOVE @fdp,a0
     00003620 00001810'
0869 00003640      8401                    MOVE *a0,A1    ;em A1 esta o Nchar field
0870 00003650      1200                    ADDK 16,a0     ;passa pelo N field
0871 00003660      0B81                    ANDi 7Fh,A1    ;pega o numero de caracteres
     00003670 FFFFFFB0
0872 00003690      4C22                    MOVE A1,A2     ;copia
0873 000036A0      1100  CREATE2:          ADDK 08,a0     ;incrementa a0 ate passar pelo string
0874 000036B0      3C42                    DSJ A2,create2
0875 000036C0      1FE1                    BTST 0,A1      ;testa se A1 e par
0876 000036D0      CA01                    JRz create3
0877 000036E0      1100                    ADDK 08,a0
0878 000036F0      4C03  CREATE3:          MOVE a0,A3     ;*A3 contem code field address
0879 00003700      1003                    ADDK 32,A3
0880 00003710      9060                    MOVE A3,*a0+
0881 00003720      0580                    MOVE a0,@fdp
     00003730 00001810'
0882 00003750      0960                    RETS           ;DP aponta p/ o code field address
0883                        *
0884                        * n CONSTANT name
0885                        *              cria no dicionario a constante name com o valor n
0886                        *
0887 00003760 00003380' H_CONSTANT:        .int H_CREATE
0888 00003780        08                    .byte 08h,0h
     00003788        00
0889 00003790        43                    .string "CONSTANT"
     00003798        4F
     000037A0        4E
     000037A8        53
     000037B0        54
     000037B8        41
     000037C0        4E
     000037C8        54
0890 000037D0                              .even
0891 000037D0 000037F0'                    .field CONSTANT
0892 000037F0      0D5F  CONSTANT:         calla create
     00003800 00003400'
0893 00003820      0D5F                    calla compile
     00003830 00001C00'
0894 00003850      0D5F                    calla doconstant
     00003860 00001170'
0895 00003880      0D5F                    calla comma
     00003890 000019B0'
0896 000038B0      0960                    RETS
0897                        *
0898                        * n VARIABLE name      cria a variavel name com o valor inicial n
0899                        *
0900 000038C0 00003760' H_VARIABLE:        .int H_CONSTANT
0901 000038E0        08                    .byte 08h,0h
     000038E8        00
0902 000038F0        56                    .string "VARIABLE"
     000038F8        41
     00003900        52
     00003908        49
     00003910        41
     00003918        42
     00003920        4C
     00003928        45
0903 00003930                              .even
0904 00003930 00003950'                    .field variable
```

```
 0905 00003950      0D5F  VARIABLE:      calla create
      00003960 00003400'
 0906 00003980      0D5F                 calla compile
      00003990 00001C00'
 0907 000039B0      0D5F                 calla docreate
      000039C0 000010A0'
 0908 000039E0      0D5F                 calla comma
      000039F0 000019B0'
 0909 00003A10      0960                 RETS
 0910                        *
 0911                        * - MODE a   variavel de controle que comtem o flag indicativo
 0912                        *            do modo corrente (interpretativo ou compilativo)
 0913                        *
 0914 00003A20 000038C0' H_MODE:        .int H_VARIABLE
 0915 00003A40        04                .byte 04h,0h
      00003A48        00
 0916 00003A50        4D                .string "MODE"
      00003A58        4F
      00003A60        44
      00003A68        45
 0917 00003A70                          .even
 0918 00003A70 00003A90'                .field MODE
 0919 00003A90      0D5F  MODE:         calla docreate
      00003AA0 000010A0'
 0920 00003AC0        00  fmode:        .byte 0h,0h,0h,0h
      00003AC8        00
      00003AD0        00
      00003AD8        00
 0921                        *
 0922                        * - : name  cria uma nova definicao no dicionario com o
 0923                        *            nome name . entra no modo compilativo
 0924                        *
 0925 00003AE0 00003A20' h_colon:       .int h_MODE
 0926 00003B00        01                .byte 01h,0h
      00003B08        00
 0927 00003B10        3A                .string ":"
 0928 00003B20                          .even
 0929 00003B20 00003B40'                .field colon
 0930 00003B40      0D5F  colon:        calla current
      00003B50 000026C0'
 0931 00003B70      0D5F                calla fetch
      00003B80 00000C10'
 0932 00003BA0      0D5F                calla context
      00003BB0 000025E0'
 0933 00003BD0      0D5F                calla store
      00003BE0 00000CB0'
 0934 00003C00      0D5F                calla create
      00003C10 00003400'
 0935 00003C30      0D5F                 calla tru
      00003C40 000013B0'
 0936 00003C60      0D5F                calla MODE
      00003C70 00003A90'
 0937 00003C90      0D5F                calla store
      00003CA0 00000CB0'
 0938 00003CC0      0960               rets
 0939                        *
 0940                        * - ; -              termina uma definicao , retornando ao modo
 0941                        *            interpretativo
 0942                        *
 0943 00003CD0 00000000  H_semi:        .int 0    ;inicio do vocabulario compiler
 0944 00003CF0        01                .byte 01h,01h
      00003CF8        01
 0945 00003D00        3B                .string ";"
 0946 00003D10                          .even
 0947 00003D10 00003D30'                .field semi
 0948 00003D30      0D5F  semi:          calla lit
      00003D40 00001230'
 0949 00003D60 00000960                 .int 0960h ;     (RETS)
 0950 00003D80      0D5F                calla ccomma
      00003D90 00001AA0'
 0951 00003DB0      0D5F                 calla fal
      00003DC0 00001470'
 0952 00003DE0      0D5F                calla mode
      00003DF0 00003A90'
 0953 00003E10      0D5F                calla store
      00003E20 00000CB0'
```

```
00003E40      0960                    RETS
                          *
                          * - <BUILDS a  usado em conjunto com DOES> . Gera um cabecalho
                          *              com a proxima palavra no TIB , insere um CALLA e
                          *              coloca o endereco da proxima celula no pilha
                          *
00003E50 00003CD0' H_builds:      .int h_semi
00003E70        07                 .byte 07,0h
00003E78        00
00003E80        3C                 .string "<BUILDS"
00003E88        42
00003E90        55
00003E98        49
00003EA0        4C
00003EA8        44
00003EB0        53
00003EC0                           .even
00003EC0 00003EE0'                 .field builds
00003EE0      0D5F  builds:       calla create
00003EF0 00003400'
00003F10      0D5F                 calla lit
00003F20 00001230'
00003F40 00000D5F                  .int 0D5Fh
00003F60      0D5F                 calla ccomma
00003F70 00001AA0'
00003F90      0D5F                 calla here
00003FA0 00001720'
00003FC0      0D5F                 calla lit
00003FD0 00001230'
00003FF0 00000002                  .int 2
00004010      0D5F                 calla allot
00004020 000018B0'
00004040      0960                 RETS
                          *
                          * a DOES> -     termina o trecho "compile time" de uma definicao e
                          *               armazena o endereco seguinte no endereco contido
                          *               na pilha , estabelecendo o trecho "run time".
                          *
00004050 00003E50' H_does:        .int h_builds
00004070        05                 .byte 05,0h
00004078        00
00004080        44                 .string "DOES>"
00004088        4F
00004090        45
00004098        53
000040A0        3E
000040B0                           .even
000040B0 000040D0'                 .field does
000040D0      95E0  does:         MOVE *SP+,a0
000040E0      A00B                 MOVE a0,-*PSTK
000040F0      0D5F                 calla swap
00004100 000001E0'
00004120      0D5F                 calla store
00004130 00000CB0'
00004150      0960                 RETS
                          *
                          * a b (FIND) a f        Rotina de busca no dicionario . Busca o
                          *                        string contido no endereco a , a partir
                          *                        do cabelhaco b . Retorna flag f=true
                          *                        e endereco a da funcao , se definida.
                          *
00004160 00003AE0' H_find:        .int H_colon
00004180        06                 .byte 06,0h
00004188        00
00004190        28                 .string "(FIND)"
00004198        46
000041A0        49
000041A8        4E
000041B0        44
000041B8        29
000041C0                           .even
000041C0 000041E0'                 .field  find
000041E0      9566  find:         MOVE *PSTK+,A6    ;First header address
000041F0      9568                 MOVE *PSTK+,A8    ;word buffer -32 (here)
00004200      4D09                 MOVE A8,A9
00004210      1008                 ADDK 32,A8
```

```
00004220      8507                    MOVE *A8,A7
00004230      0B87                    ANDi 0FFh, A7     ;separa Nchar (buffer)
00004240 FFFFFF00
00004260      CA28                    JRZ F_NAoaCHOU
00004270      1208                    ADDk 16,A8        ;Name field (buffer)
00004280      54C6                    oR A6,A6
00004290      CA25                    JRz F_naoachou
000042A0      84C5  F_COMPARA:        MOVE *A6,A5
000042B0      1006                    ADDk 32,A6        ;Nchar field address fetch ic
000042C0      84C4                    MOVE *A6,A4       ;Nchar
000042D0      4C8A                    MOVE A4,A10
000042E0      0B8A                    ANDi 0FFh,A10     ;Nchar (Dic)
000042F0 FFFFFF00
00004310      4947                    CMP A10,A7        ;compara comprimentos
00004320      CB18                    JRNE F_proximo    ;se   pego o proximo
00004330      4D03                    MOVE A8,A3        ;A3 e o pointer no buffer
00004340      5642                    clr A2        ;A6 pointer no Dic
00004350      4C41                    MOVE A2,A1
00004360      1206                    addk 16,a6
00004370      8EC2  F_COMPCHAR:       MOVB *A6,A2       ;compara os caracteres
00004380      8E61                    MOVB *A3,A1
00004390      4841                    CMP A2,A1
000043A0      CB10                    JRNE F_proximo
000043B0      1106                    ADDk 8,A6
000043C0      1103                    ADDk 8,A3
000043D0      3CEA                    DSJS A10,F_COMPCHAR
000043E0      1FE4  F_ACHOU:          Btst 0,A4         ;testa se e impar
000043F0      CA01                    JRZ F_ACHOU1
00004400      1106                    ADDK 8,A6
00004410      84C0  F_ACHOU1:         MOVE *A6,a0
00004420      A00B                    MOVE a0,-*PSTK    ;salva CFA na pilha
00004430      09E0                    MOVi TRUE,a0
00004440 FFFFFFFF
00004460      1EE4                    Btst 8,A4         ;testa se e imediate
00004470      CA01                    JRZ F_ACHOU2
00004480      03A0                    NEG a0
00004490      A00B  F_ACHOU2:         MOVE a0,-*PSTK    ;se -1 normal/ se 1 imediate
000044A0      C008                    JRUC F_SAI
000044B0      54A5  F_PROXIMO:        OR A5,A5
000044C0      CA02                    JRZ F_NaoACHOU
000044D0      4CA6                    MOVE A5,A6
000044E0      C0DB                    JRUC F_COMPARA
000044F0      A12B  F_NAOACHOU:       MOVE A9,-*PSTK
00004500      09C0                    MOVi 0,A0
00004510      0000
00004520      A00B                    MOVE A0,-*PSTK
00004530      0960  F_SAI:            RETS
                    *
                    * - IMMEDIATE -          pos o atributo de imediato na ultima
                    *                         definicao efetudada
                    *
00004540 00004160' H_IMMEDIATE:       .int H_find
00004560        09                    .byte 09h,0h
00004568        00
00004570        49                    .string "IMMEDIATE"
00004578        4D
00004580        4D
00004588        45
00004590        44
00004598        49
000045A0        41
000045A8        54
000045B0        45
000045C0                              .even
000045C0 000045E0'                    .field IMMEDIATE
000045E0      0D5F  IMMEDIATE:        calla last
000045F0 00002BA0'
00004610      0D5F                    calla fetch
00004620 00000C10'
00004640      9560                    MOVE *PSTK+,a0
00004650      1000                    ADDK 32,a0
00004660      8401                    MOVE *a0,A1
00004670      0BA1                    ori 100h,A1           ;seta precedente bit
00004680 00000100
000046A0      8020                    MOVE A1, *a0
000046B0      0960                    RETS
```

```
                        *
                        * - (") a n     retorna na pilha o endereco e o comprimento de
                        *               um string colocado nas proximas celulas no
                        *               dicionario
                        *
000046C0 00004540' H_rquote:          .int H_immediate
000046E0       03                    .byte 03,0h
000046E8       00
000046F0       28                    .string "("
000046F8       22                   .byte 22h,29h
00004700       29
00004710                             .even
00004710 00004730'                   .field rquote
00004730     85E0  rquote:              MOVE *SP,a0      ;a0 contem endereco de retorno
00004740     8401                    MOVE *a0,A1      ;A1 contem o Nchar field
00004750     0B81                    ANDi 0FFh,a1        ;contem o Nchar
00004760 FFFFFF00
00004780     4C22                    MOVE A1,A2
00004790     1200                    ADDk 16,a0       ;a0 aponta para o inicio do
000047A0     A00B                    MOVE a0,-*PSTK   ;poe pilha
000047B0     A04B                    MOVE A2,-*PSTK   ;poe comprimento
000047C0     1FE1                    BTST 0,A1
000047D0     CA01                    JRZ rquote1
000047E0     1021                    ADDK 1 ,A1
000047F0     2461  rquote1:             SLL 3,A1         ;passa pelo string
00004800     4020                    ADD A1,a0
00004810     800F                    MOVE a0,*SP
00004820     0960                    RETS
                        *
                        * - ," string "                 compila o string no dicionario
                        *
00004830 000046C0' H_commaquote: .int H_rquote
00004850       02                    .byte 02,0h
00004858       00
00004860       2C                    .string  ","
00004868       22                   .byte 22h
00004870                             .even
00004870 00004890'                   .field commaquote
00004890     09C0  commaquote:       MOVi 22h,a0
000048A0     0022
000048B0     A00B                    MOVE a0,-*PSTK
000048C0     05A2                    MOVE @fdp,A2
000048D0 00001810'
000048F0     1402                    SUBK 32,A2           ;alinha @DP
00004900     0582                    MOVE A2,@fdp
00004910 00001810'
00004930     0D5F                    calla token           ;poe o string no word buffer
00004940 00002FA0'
00004960     05A3                    MOVE @fdp,A3
00004970 00001810'
00004990     8461                    MOVE *A3,A1
000049A0     0B81                    ANDi 0FFh,a1            ;pega Nchar
000049B0 FFFFFF00
000049D0     4C22                    MOVE A1,A2
000049E0     1200                    ADDK 16,a0
000049F0     1FE1                    BTST 0,A1
00004A00     CA01                    JRZ commaquote1               ;conta numero bits
00004A10     1021                    ADDK 1,A1
00004A20     2461  commaquote1:      SLL 3,A1
00004A30     4023                    ADD A1,A3
00004A40     0583                    MOVE A3,@fdp          ;atualiza DP
00004A50 00001810'
00004A70     0960                    RETS
                        *
                        * - ." string "                 compila o string no dicionario e insere a
                        *                        funcao (.")
                        *
00004A80 00004830' H_rdotquote:  .int H_commaquote
00004AA0       04                    .byte 04h,0h
00004AA8       00
00004AB0       28                    .string "(."
00004AB8       2E
00004AC0       22                   .byte 22h,29h
00004AC8       29
00004AD0                             .even
00004AD0 00004AF0'                   .field rdotquote
```

```
00004AF0      85E0  rdotquote:    MOVE *SP,a0
00004B00      8401                MOVE *a0,A1
00004B10      0B81                ANDi 0FFh,a1
00004B20 FFFFFF00
00004B40      4C22                MOVE A1,A2
00004B50      1200                ADDK 16,a0
00004B60      A00B                MOVE a0,-*PSTK
00004B70      A04B                MOVE A2,-*PSTK
00004B80      1FE1                BTST 0,A1
00004B90      CA01                JRZ rdotquotel
00004BA0      1021                ADDk 1,A1
00004BB0      2461  rdotquotel:   SLL 3,A1
00004BC0      4020                ADD A1,a0
00004BD0      800F                MOVE a0,*SP
00004BE0      0D5F                calla type
00004BF0 00002410'
00004C10      0960                rets
00004C20 00004050' H_dotquote:    .int H_does
00004C40        02                .byte 02h,1h
00004C48        01
00004C50        2E                .string ".*
00004C58        22                .byte 22h
00004C60                          .even
00004C60 00004C80'                .field dotquote
00004C80      0D5F  dotquote:     calla compile
00004C90 00001C00'
00004CB0      0D5F                calla rdotquote
00004CC0 00004AF0'
00004CE0      0D5F                calla commaquote
00004CF0 00004890'
00004D10      0960                RETS
                    *
                    * - " string "        compila o string no dicionario inserindo a
                    *                     funcao (")
                    *
00004D20 00004C20' H_quote:       .int H_dotquote
00004D40        02                .byte 02,01h
00004D48        01
00004D50        22                .byte 22h
00004D60                          .even
00004D60 00004D80'                .field quote
00004D80      0D5F  quote:        calla compile
00004D90 00001C00'
00004DB0      0D5F                calla rquote
00004DC0 00004730'
00004DE0      0D5F                calla commaquote
00004DF0 00004890'
00004E10      0960                RETS
                    *
                    * x y * z      multiplicacao
                    *
00004E20 00004A80' H_times:       .int H_rdotquote
00004E40        01                .byte 01h,0h
00004E48        00
00004E50        2A                .string "*"
00004E60                          .even
00004E60 00004E80'                .field times
00004E80      9560  times:        MOVE *PSTK+,a0
00004E90      9561                MOVE *PSTK+,A1
00004EA0      018A                getst A10
00004EB0      0740                SETF 32,0,1
00004EC0      5C01                MPYS a0,A1
00004ED0      01AA                PUTST A10
00004EE0      A02B                MOVE A1,-*PSTK
00004EF0      0960                RETS
                    *
                    * x y / z      divisao x/y
                    *
00004F00 00004E20' H_div:         .int H_times
00004F20        01                .byte 01h,0h
00004F28        00
00004F30        2F                .string "/"
00004F40                          .even
00004F40 00004F60'                .field div
00004F60      9560  div:          MOVE *PSTK+,a0    ;(X)
00004F70      9563                MOVE *PSTK+,A3    ;(Y)
```

```
00004F80      018A                getst a10
00004F90      0740                setf 32,0,1
00004FA0      5803                 DIVS a0,A3          ;Y/X
00004FB0      01AA                putst a10
00004FC0      A06B                 MOVE A3,-*PSTK
00004FD0      0960                 RETS
                      *
                      * x y MOD z     resto da divisao x/y
                      *
00004FE0 00004F00' H_MOD:          .int H_div
00005000        03                 .byte 03h,0h
00005008        00
00005010        4D                 .string "MOD"
00005018        4F
00005020        44
00005030                           .even
00005030 00005050'                 .field MOD
00005050      9560  MOD:           MOVE  *PSTK+,a0     ; (X)
00005060      9562                 MOVE  *PSTK+,A2     ; (Y)
00005070      018A                  getst a10
00005080      0540                  setf 32,0,0
00005090      6C02                 MODS  A0,A2
000050A0      01AA                  putst a10
000050B0      A04B                 MOVE A2,-*PSTK      ; resto
000050C0      0960                 RETS
                      *
                      * x y /MOD r q                    divisao x/y apresentando quociente e resto
                      *
000050D0 00004FE0' H_divMOD:       .int H_MOD
000050F0        04                 .byte 04,0h
000050F8        00
00005100        2F                 .string "/MOD"
00005108        4D
00005110        4F
00005118        44
00005120                           .even
00005120 00005140'                 .field divMOD
00005140      9560  divMOD:        MOVE *PSTK+,a0      ;  (X)
00005150      9563                 MOVE *PSTK+,A3        ; (Y)
00005160      018A                  getst a10
00005170      0740                  setf 32,0,1
00005180      5642                 CLR A2
00005190      1C03                 BTST 31,A3
000051A0      CA01                 JRZ  divMOD1
000051B0      03E2                 NOT A2
000051C0      5802  divMOD1:       DIVS a0,A2
000051D0      01AA                  putst a10
000051E0      A04B                 MOVE A2,-*PSTK        ;(quociente)
000051F0      A06B                 MOVE A3,-*PSTK        ;(resto)
00005200      0960                 RETS
                      *
                      * x y z */  q          multiplica xy e divide por z apresentando
                      *                      quociente
                      *
00005210 000050D0' H_timesdiv:          .int H_divMOD
00005230        02                 .byte 02,0h
00005238        00
00005240        2A                 .string "*/"
00005248        2F
00005250                           .even
00005250 00005270'                 .field timesdiv
00005270      9564  timesdiv:      MOVE *PSTK+, A4        ;n3
00005280      9562                 MOVE *PSTK+, A2        ;n2
00005290      9560                 MOVE *PSTK+, a0        ;n1
000052A0      018A                 GETST a10
000052B0      0740                 SETF 32,0,1
000052C0      5C02                 MPYS a0,A2             ;n1 x n2
000052D0      5882                 DIVS A4,A2             ;n1 x n2 / n3
000052E0      01AA                 PUTST A10
000052F0      A04B                 MOVE A2,-*PSTK
00005300      0960                 RETS
                      *
                      * x y z */MOD r q      multiplica xy e divide por z apresentando
                      *                      quociente e resto
                      *
00005310 00005210' H_timesdivMOD: .int H_timesdiv
```

```
00005330        05                      .byte 05h,0h
00005338        00
00005340        2A                      .string "*/MOD"
00005348        2F
00005350        4D
00005358        4F
00005360        44
00005370                                .even
00005370 00005390'                      .field timesdivMOD
00005390      9564  timesdivMOD:        MOVE *PSTK+, A4       ;n3
000053A0      9562                      MOVE *PSTK+, A2       ;n2
000053B0      9560                      MOVE *PSTK+,a0        ;n1
000053C0      018A                      GETST a10
000053D0      0740                      SETF 32,0,1
000053E0      5C02                      MPYS a0,A2
000053F0      5882                      DIVS A4,A2
00005400      01AA                      PUTST A10
00005410      A04B                      MOVE A2,-*PSTK        ;quociente
00005420      A06B                      MOVE A3,-*PSTK        ;resto
00005430      0960                      RETS
                            *
                            * x ABS y      valor absoluto
                            *
00005440 00005310'  h_abs:              .int H_timesdivMOD
00005460        03                      .byte 03h,0h
00005468        00
00005470        41                      .string "ABS"
00005478        42
00005480        53
00005490                                .even
00005490 000054B0'                      .field ABS
000054B0      9560  ABS:                MOVE *PSTK+,A0
000054C0      0380                      ABS A0
000054D0      A00B                      MOVE A0,-*PSTK
000054E0      0960                      RETS
                            *
                            * x NEGATE y   complemento de dois
                            *
000054F0 00005440'  H_NEGATE:           .int H_ABS
00005510        06                      .byte 06h,0h
00005518        00
00005520        4E                      .string "NEGATE"
00005528        45
00005530        47
00005538        41
00005540        54
00005548        45
00005550                                .even
00005550 00005570'                      .field NEGATE
00005570      9560  NEGATE:             MOVE *PSTK+,A0
00005580      03A0                      NEG A0
00005590      A00B                      MOVE A0,-*PSTK
000055A0      0960                      RETS
                            *
                            * x y MAX z     retorna o maior entre os dois numeros
                            *
000055B0 000054F0'  H_MAX:              .int H_NEGATE
000055D0        03                      .byte 03h,0h
000055D8        00
000055E0        4D                      .string "MAX"
000055E8        41
000055F0        58
00005600                                .even
00005600 00005620'                      .field MAX
00005620      9560  MAX:                MOVE *PSTK+,A0
00005630      9561                      MOVE *PSTK+,A1
00005640      4820                      CMP A1,A0
00005650      C701                      JRGT MAX2
00005660      4C20                      MOVE A1,A0
00005670      A00B  MAX2:               MOVE A0, -*PSTK
00005680      0960                      RETS
                            *
                            * x y MIN z     retorna o menor entre dois numeros
                            *
00005690 000055B0'  H_MIN:              .int H_MAX
000056B0        03                      .byte 03h,0h
```

```
      000056B8        00
 1316 000056C0        4D                    .string "MIN"
      000056C8        49
      000056D0        4E
 1317 000056E0                              .even
 1318 000056E0 00005700'                    .field MIN
 1319 00005700      9560  MIN:              MOVE *PSTK+,A0
 1320 00005710      9561                    MOVE *PSTK+,A1
 1321 00005720      4820                    CMP A1,A0
 1322 00005730      C401                    JRLT MIN2
 1323 00005740      4C20                    MOVE A1,A0
 1324 00005750      A00B  MIN2:             MOVE A0,-*PSTK
 1325 00005760      0960                    RETS
 1326                     *
 1327                     * - >MARK a      reserva uma celula vazia no dicionario e poe o
 1328                     *                 endereco na pilha
 1329                     *
 1330 00005770 00004D20' H_fmark:           .int H_quote
 1331 00005790        05                    .byte 05h,0h
      00005798        00
 1332 000057A0        3E                    .string ">MARK"
      000057A8        4D
      000057B0        41
      000057B8        52
      000057C0        4B
 1333 000057D0                              .even
 1334 000057D0 000057F0'                    .field fmark
 1335 000057F0      0D5F  fmark:            calla HERE
      00005800 00001720'
 1336 00005820      0D5F                    calla fal
      00005830 00001470'
 1337 00005850      0D5F                    calla comma
      00005860 000019B0'
 1338 00005880      0960                    RETS
 1339                     *
 1340                     * a >RESOLVE -          poe o endereco da proxima celula do dicionario
 1341                     *                 na celula apontada pelo endereco fornecido
 1342                     *
 1343 00005890 00005770' H_fresolve:        .int H_fmark
 1344 000058B0        08                    .byte 08h,0h
      000058B8        00
 1345 000058C0        3E                    .string ">RESOLVE"
      000058C8        52
      000058D0        45
      000058D8        53
      000058E0        4F
      000058E8        4C
      000058F0        56
      000058F8        45
 1346 00005900                              .even
 1347 00005900 00005920'                    .field fresolve
 1348 00005920      0D5F  fresolve:         calla HERE
      00005930 00001720'
 1349 00005950      0D5F                    calla swap
      00005960 000001E0'
 1350 00005980      0D5F                    calla store
      00005990 00000CB0'
 1351 000059B0      0960                    RETS
 1352                     *
 1353                     * - <MARK a      poe o endereco da proxima celula do dicionario
 1354                     *                 na pilha
 1355                     *
 1356 000059C0 00005890' H_bmark:             .int H_fresolve
 1357 000059E0        05                    .byte 05h,0h
      000059E8        00
 1358 000059F0        3C                    .string "<MARK"
      000059F8        4D
      00005A00        41
      00005A08        52
      00005A10        4B
 1359 00005A20                              .even
 1360 00005A20 00005A40'                    .field bmark
 1361 00005A40      0D5F  bmark:            calla HERE
      00005A50 00001720'
 1362 00005A70      0960                    RETS
 1363                     *
```

```
                                * a <RESOLVE - compila o endereco fornecido , na proxima
                                *               celula do dicionario
                                *
00005A80 000059C0' H_bresolve:   .int H_bmark
00005AA0       08                .byte 08h,0h
00005AA8       00
00005AB0       3C                .string "<RESOLVE"
00005AB8       52
00005AC0       45
00005AC8       53
00005AD0       4F
00005AD8       4C
00005AE0       56
00005AE8       45
00005AF0                         .even
00005AF0 00005B10'               .field bresolve
00005B10     0D5F  bresolve:     calla comma
00005B20 000019B0'
00005B40     0960                RETS
                                *
                                * - IF a        monta o comando ?BRANC no dicionario e executa
                                *               >MARK
                                *
00005B50 00005A80' H_IF:         .int H_bresolve
00005B70       02                .byte 02h,01h
00005B78       01
00005B80       49                .string "IF"
00005B88       46
00005B90                         .even
00005B90 00005BB0'               .field IF
00005BB0     0D5F  IF:           calla COMPILE
00005BC0 00001C00'
00005BE0     0D5F                calla cbranch
00005BF0 00001600'
00005C10     0D5F                calla fmark
00005C20 000057F0'
00005C40     0960                RETS
                                *
                                * a ELSE a      monta o comando BRANCH no dicionario , executa
                                *               a montagem do jumps necessarios
                                *
00005C50 00005B50' H_ELSE:       .int H_IF
00005C70       04                .byte 04h,01h
00005C78       01
00005C80       45                .string "ELSE"
00005C88       4C
00005C90       53
00005C98       45
00005CA0                         .even
00005CA0 00005CC0'               .field ELSE
00005CC0     0D5F  ELSE:         calla COMPILE
00005CD0 00001C00'
00005CF0     0D5F                calla BRANCH
00005D00 00001530'
00005D20     0D5F                calla fmark
00005D30 000057F0'
00005D50     0D5F                calla SWAP
00005D60 000001E0'
00005D80     0D5F                calla fresolve
00005D90 00005920'
00005DB0     0960                RETS
                                *
                                * a THEN -      executa >RESOLVE
                                *
00005DC0 00005C50' H_THEN:       .int H_ELSE
00005DE0       04                .byte 04h,01h
00005DE8       01
00005DF0       54                .string "THEN"
00005DF8       48
00005E00       45
00005E08       4E
00005E10                         .even
00005E10 00005E30'               .field THEN
00005E30     0D5F  THEN:         calla fresolve
00005E40 00005920'
00005E60     0960                RETS
```

```
                                *
                                * - BEGIN a     executa <MARK
                                *
00005E70 00005DC0' H_BEGIN:      .int H_THEN
00005E90        05               .byte 05h,01h
00005E98        01
00005EA0        42               .string "BEGIN"
00005EA8        45
00005EB0        47
00005EB8        49
00005EC0        4E
00005ED0                         .even
00005ED0 00005EF0'               .field BEGIN
00005EF0      0D5F  BEGIN:       calla bmark
00005F00 00005A40'
00005F20      0960               RETS
                                *
                                * a UNTIL -     monta no dicionario ?BRANCH e executa <RESOLVE
                                *
00005F30 00005E70' H_UNTIL:      .int H_BEGIN
00005F50        05               .byte 05h,01h
00005F58        01
00005F60        55               .string "UNTIL"
00005F68        4E
00005F70        54
00005F78        49
00005F80        4C
00005F90                         .even
00005F90 00005FB0'               .field UNTIL
00005FB0      0D5F  UNTIL:       calla COMPILE
00005FC0 00001C00'
00005FE0      0D5F               calla cbranch
00005FF0 00001600'
00006010      0D5F               calla bresolve
00006020 00005B10'
00006040      0960               RETS
00006050 00005F30' H_AGAIN:       .int H_UNTIL
00006070        05                .byte 05h,0h
00006078        00
00006080        41                .string "AGAIN"
00006088        47
00006090        41
00006098        49
000060A0        4E
000060B0                          .even
000060B0 000060D0'                .field again
000060D0      0D5F  again:        calla compile
000060E0 00001C00'
00006100      0D5F                calla branch
00006110 00001530'
00006130      0D5F                calla bresolve
00006140 00005B10'
00006160      0960                rets
                                *
                                * - WHILE a     monta o comando ?BRANCH no dic. e executa >MARK
                                *
00006170 00006050' H_WHILE:      .int H_again
00006190        05               .byte 05h,01h
00006198        01
000061A0        57               .string "WHILE"
000061A8        48
000061B0        49
000061B8        4C
000061C0        45
000061D0                         .even
000061D0 000061F0'               .field WHILE
000061F0      0D5F  WHILE:       calla COMPILE
00006200 00001C00'
00006220      0D5F               calla cbranch
00006230 00001600'
00006250      0D5F               calla fmark
00006260 000057F0'
00006280      0960               RETS
                                *
                                * a REPEAT a    monta o comando BRANCH resolvendo os jumps
                                *                necessarios
```

```
00006290 00006170' H_REPEAT:      .int H_WHILE
000062B0        06                .byte 06h,01h
000062B8        01
000062C0        52                .string "REPEAT"
000062C8        45
000062D0        50
000062D8        45
000062E0        41
000062E8        54
000062F0                          .even
000062F0 00006310'                .field REPEAT
00006310      0D5F  REPEAT:       calla SWAP
00006320 000001E0'
00006340      0D5F                calla COMPILE
00006350 00001C00'
00006370      0D5F                calla BRANCH
00006380 00001530'
000063A0      0D5F                calla bresolve
000063B0 00005B10'
000063D0      0D5F                calla fresolve
000063E0 00005920'
00006400      0960                RETS
                        *
                        * - DO a    monta no dic. a funcao (DO) e executa >MARK
                        *
00006410 00006290' H_DO:          .int H_REPEAT
00006430        02                .byte 02h,01h
00006438        01
00006440        44                .string "DO"
00006448        4F
00006450                          .even
00006450 00006470'                .field DO
00006470      0D5F  DO:           calla COMPILE
00006480 00001C00'
000064A0      0D5F                calla rdo
000064B0 00001D50'
000064D0      0D5F                calla fmark
000064E0 000057F0'
00006500      0960                RETS
                        *
                        * a LOOP -  monta no dic. a funcao (LOOP) , soma 32 em a e
                        *           resolve os jumps
                        *
00006510 00006410' H_LOOP:        .int H_DO
00006530        04                .byte 04h,01h
00006538        01
00006540        4C                .string "LOOP"
00006548        4F
00006550        4F
00006558        50
00006560                          .even
00006560 00006580'                .field LOOP
00006580      0D5F  LOOP:         calla COMPILE
00006590 00001C00'
000065B0      0D5F                calla rloop
000065C0 00001E70'
000065E0      0D5F                calla DUP
000065F0 000000A0'
00006610      0D5F                calla lit
00006620 00001230'
00006640 00000020                 .INT 32
00006660      0D5F                calla plus
00006670 00002890'
00006690      0D5F                calla bresolve
000066A0 00005B10'
000066C0      0D5F                calla fresolve
000066D0 00005920'
000066F0      0960                RETS
                        *
                        * a +LOOP -  idem a LOOP montando (+LOOP)
                        *
00006700 00006510' H_plusLOOP:    .int H_LOOP
00006720        05                .byte 05h,01h
00006728        01
00006730        2B                .string "+LOOP"
00006738        4C
```

```
      00006740        4F
      00006748        4F
      00006750        50
1505  00006760                                  .even
1506  00006760 00006780'                        .field plusloop
1507  00006780      0D5F  plusloop:             calla COMPILE
      00006790 00001C00'
1508  000067B0      0D5F                        calla rplusloop
      000067C0 00002000'
1509  000067E0      0D5F                        calla DUP
      000067F0 000000A0'
1510  00006810      0D5F                        calla lit
      00006820 00001230'
1511  00006840 00000020                        .int 32
1512  00006860      0D5F                        calla plus
      00006870 00002890'
1513  00006890      0D5F                        calla bresolve
      000068A0 00005B10'
1514  000068C0      0D5F                        calla fresolve
      000068D0 00005920'
1515  000068F0      0960                        RETS
1516                          *
1517                          * - LEAVE -        monta no dic. a funcao RLEAVE
1518                          *
1519  00006900 00006700' H_LEAVE:               .int H_plusloop
1520  00006920        05                        .byte 05h,01h
      00006928        01
1521  00006930        4C                        .string "LEAVE"
      00006938        45
      00006940        41
      00006948        56
      00006950        45
1522  00006960                                  .even
1523  00006960 00006980'                        .field LEAVE
1524  00006980      0D5F  LEAVE:                calla COMPILE
      00006990 00001C00'
1525  000069B0      0D5F                        calla rleave
      000069C0 00002280'
1526  000069E0      0960                        RETS
1527                          *
1528                          * - ?LEAVE -       monta no dic. a funcao RCLEAVE
1529                          *
1530  000069F0 00006900' H_CLEAVE:               .int H_LEAVE
1531  00006A10        06                        .byte 06h,01h
      00006A18        01
1532  00006A20        3F                        .string "?LEAVE"
      00006A28        4C
      00006A30        45
      00006A38        41
      00006A40        56
      00006A48        45
1533  00006A50                                  .even
1534  00006A50 00006A70'                        .field CLEAVE
1535  00006A70      0D5F  CLEAVE:               calla COMPILE
      00006A80 00001C00'
1536  00006AA0      0D5F                        calla rcleave
      00006AB0 00002360'
1537  00006AD0      0960                        RETS
1538                          *
1539                          * - BASE a         variavel de controle , armazena a base
1540                          *                  numerica corrente
1541                          *
1542  00006AE0 00005690' H_BASE:                 .int H_MIN
1543  00006B00        04                        .byte 04h,01h
      00006B08        01
1544  00006B10        42                        .string "BASE"
      00006B18        41
      00006B20        53
      00006B28        45
1545  00006B30                                  .even
1546  00006B30 00006B50'                        .field BASE
1547  00006B50      0D5F  BASE:                 calla DOCREATE
      00006B60 000010A0'
1548  00006B80 0000000A  fBASE:                 .int 10
1549                          *
1550                          * - NUMBER x f converte o string no word-buffer num numero
```

```
                        *                 segundo a base numerica corrente . f=true sucesso
                        *
00006BA0 00006AE0' H_number:    .int H_BASE
00006BC0        08                .byte 08h,0h
00006BC8        00
00006BD0        6E                .string "number"
00006BD8        75
00006BE0        6D
00006BE8        62
00006BF0        65
00006BF8        72
00006C00                          .even
00006C00 00006C20'                .field number
00006C20      05AA  number:     MOVE @fdp,A10
00006C30 00001810'
00006C50      0184                GETST A4
00006C60      0740                SETF 32, 0, 1
00006C70      05A2                MOVE @fBASE,A2
00006C80 00006B80'
00006CA0      4C43                MOVE A2, A3
00006CB0      0BE3                SUBI 01, A3
00006CC0      FFFE
00006CD0      100A                ADDK 32, A10
00006CE0      8549                MOVE *A10, A9    ;A10 aponta para nchar field
00006CF0      0B89                ANDI 0FFh, A9    ;A9 counter NCHAR
00006D00 FFFFFF00
00006D20      120A                ADDK 16, A10     ;A10 aponta para name field
00006D30      5708                CLR A8           ;Flag = false
00006D40      56E7                CLR A7           ;Flag sinal = 0
00006D50      5600                CLR A0           ;NCHAR = 0
00006D60      5621                CLR A1           ;Result n mero = 0
00006D70      0B49                CMPI 0,A9        ;Teste de NCHAR = 0
00006D80      FFFF
00006D90      CA20                JRZ numberSAI1
00006DA0      8F40                MOVb *A10,A0
00006DB0      0B40                CMPI 2dh,A0
00006DC0      FFD2
00006DD0      CB04                JRNE number1
00006DE0      110A                ADDK 8,A10
00006DF0      0BE9               subi 1,a9
00006E00      FFFE
00006E10      03E7                not A7           ;Sinal = -1 (negativo)
00006E20      8F40  number1:    MOVb *A10,A0
00006E30      0BE0                SUBI 30h,A0
00006E40      FFCF
00006E50      CE14                JRN numberSai1 ;Sai se caracter invalidolido
00006E60      0B40                CMPI 09,A0       ;        (>()
00006E70      FFF6
00006E80      C605                JRLE number2
00006E90      0B40                CMPI 011h,A0     ;        (>11h)
00006EA0      FFEE
00006EB0      C60E                JRLE numbersai1 ;Sai se caracter invalidolido
00006EC0      0BE0                SUBI 7,A0
00006ED0      FFF8
00006EE0      4803  number2:      CMP A0,A3           ;(num<BASE-1)
00006EF0      C40A                JRLT numberSai1  ;  Sai se caracter invalidolido
00006F00      5C41                MPYS A2,A1
00006F10      4001                ADD A0,A1
00006F20      110A                ADDK 8,A10
00006F30      3E49                DSJ A9,number1
00006F40      0B47                CMPI 0,A7
00006F50      FFFF
00006F60      CA01                JRZ number3
00006F70      03A1                NEG A1
00006F80      03E8  number3:    not A8      ; True em A8, sucesso na conversao
00006F90      A02B                MOVE A1,-*PSTK  ;Coloca na pilha o resultado
00006FA0      A10B  numberSAI1: MOVE A8,-*PSTK  ;Coloca flag
00006FB0      01A4                PUTST A4
00006FC0      0960                RETS
                        *
                        * - ASCII c     compila o proximo caracter no dic. No run-time
                        *               coloca na pilha
                        *
00006FD0 000069F0' H_ASCII:        .int H_cleave
00006FF0        05                 .byte 05h,01h
00006FF8        01
```

```
00007000        41                      .string "ASCII"
00007008        53
00007010        43
00007018        49
00007020        49
00007030                                .even
00007030 00007050'                      .field ASCII
00007050      0D5F  ASCII:              calla spc
00007060 00001300'
00007080      0D5F                      calla token
00007090 00002FA0'
000070B0      0D5F                      calla DIP
000070C0 000017E0'
000070E0      0D5F                      calla lit
000070F0 00001230'
00007110 00000030                       .int 48
00007130      0D5F                      calla plus
00007140 00002890'
00007160      0D5F                      calla fetch
00007170 00000C10'
00007190      0D5F                      calla tru
000071A0 000013B0'
000071C0      0D5F                      calla AND
000071D0 000004E0'
000071F0      0D5F                      calla Compile
00007200 00001C00'
00007220      0D5F                      calla lit
00007230 00001230'
00007250      0D5F                      calla dot
00007260 000085A0'
00007280      0960                      RETS
                      *
                      * - HLD a         variavel de controle , armazena o ponteiro em PAD
                      *
00007290 00006BA0'    H_HLD:            .int H_number
000072B0        03                      .byte 03,0h
000072B8        00
000072C0        48                      .string "HLD"
000072C8        4C
000072D0        44
000072E0                                .even
000072E0 00007300'                      .field HLD
00007300      0D5F  HLD:                calla DOCREATE
00007310 000010A0'
00007330 00000000                       .int 0
                      *
                      * - PAD a         variavel de controle , contendo o endereco inicial
                      *                 do buffer de rascunho PAD
                      *
00007350 00007290'    H_PAD:            .int H_HLD
00007370        03                      .byte 03,0h
00007378        00
00007380        50                      .string "PAD"
00007388        41
00007390        44
000073A0                                .even
000073A0 000073C0'                      .field PAD
000073C0      0D5F  PAD:                calla DOCREATE
000073D0 000010A0'
000073F0 00000000     fpad:             .int 0
                      *
                      * - >OUT a        variavel de controle , contendo o ponteiro em TOB
                      *
00007410 00007350'    H_gOUT:                   .int H_PAD
00007430        04                      .byte 04h,0h
00007438        00
00007440        3E                      .string ">OUT"
00007448        4F
00007450        55
00007458        54
00007460                                .even
00007460 00007480'                      .field gOUT
00007480      0D5F  gOUT:               calla docreate
00007490 000010A0'
000074B0 00000000     ftobp:            .int 0
                      *
```

```
                                * - TIB a       variavel de controle , contendo o endereco inicial
                                *               do buffer de entrada
                                *
00007400 00007410' H_TIB:       .int H_gOUT
000074F0        03              .byte 03h,0h
000074F8        00
00007500        54              .string "TIB"
00007508        49
00007510        42
00007520                        .even
00007520 00007540'              .field TIB
00007540      0D5F  TIB:        calla docreate
00007550 000010A0'
00007570 00000000  fTIB:        .int 0
                                *
                                * - >IN a       variavel de controle , contendo o ponteiro em TIB
                                *
00007590 000074D0' H_gIN:       .int H_TIB
000075B0        03              .byte 03h,0h
000075B8        00
000075C0        3E              .string ">IN"
000075C8        49
000075D0        4E
000075E0                        .even
000075E0 00007600'              .field gIN
00007600      0D5F  gIN:        calla docreate
00007610 000010A0'
00007630 00000000  finp:        .int 0
                                *
                                * c HOLD -      insere o caracter c no string sendo montado em PAD
                                *
00007650 00007590' H_HOLD:      .int H_gIN
00007670        04              .byte 04,0h
00007678        00
00007680        48              .string "HOLD"
00007688        4F
00007690        4C
00007698        44
000076A0                        .even
000076A0 000076C0'              .field HOLD
000076C0      0D5F  HOLD:       calla lit
000076D0 00001230'
000076F0 FFFFFFF8                .int -8
00007710      0D5F              calla HLD
00007720 00007300'
00007740      0D5F              calla plustore
00007750 00000EF0'
00007770      0D5F              calla HLD
00007780 00007300'
000077A0      0D5F              calla fetch
000077B0 00000C10'
000077D0      0D5F              calla cstore
000077E0 00000E20'
00007800      0960              RETS
                                *
                                * - <# -        inicializa a transformacao de um numero em string
                                *               prepara PAD
                                *
00007810 00007290' H_ltsharp:   .int H_HLD
00007830        02              .byte 02,0h
00007838        00
00007840        3C              .string "<#"
00007848        23
00007850                        .even
00007850 00007870'              .field ltsharp
00007870      0D5F  ltsharp:    calla HERE
00007880 00001720'
000078A0      0D5F              calla lit
000078B0 00001230'
000078D0 000008A0               .int 69*32
000078F0      0D5F              calla plus
00007900 00002890'
00007920      0D5F              calla dup
00007930 000000A0'
00007950      0D5F              calla PAD
00007960 000073C0'
```

```
00007980      0D5F                calla store
00007990 00000CB0'
00079B0      0D5F                calla HLD
000079C0 00007300'
000079E0      0D5F                calla store
000079F0 00000CB0'
00007A10      0960                RETS
                        *
                        * x # y            converte um digito de x , coloca o caracter em PAD
                        *           e retorna o numero restante na pilha
                        *
00007A20 00007810' H_sharp:       .field H_ltsharp
00007A40        01                .byte 01,0h
00007A48        00
00007A50        23                .string "#"
00007A60                          .even
00007A60 00007A80'                .field sharp
00007A80      0D5F  sharp:        calla base
00007A90 00006B50'
00007AB0      0D5F                calla fetch
00007AC0 00000C10'
00007AE0      0D5F                calla divMOD
00007AF0 00005140'
00007B10      0D5F                calla ROT
00007B20 00000350'
00007B40      0D5F                calla lit
00007B50 00001230'
00007B70 00000009                 .int 9
00007B90      0D5F                calla over
00007BA0 000002A0'
00007BC0      0D5F                calla lt
00007BD0 00000700'
00007BF0      0D5F                calla cbranch
00007C00 00001600'
00007C20 00007CC0'                .field sharpl
00007C40      0D5F                calla lit
00007C50 00001230'
00007C70 00000007                 .int 7
00007C90      0D5F                calla plus
00007CA0 00002890'
00007CC0      0D5F  sharpl:       calla lit
00007CD0 00001230'
00007CF0        30                .byte "0",0,0,0
00007CF8        00
00007D00        00
00007D08        00
00007D10      0D5F                calla plus
00007D20 00002890'
00007D40      0D5F                calla HOLD
00007D50 000076C0'
00007D70      0960                RETS
                        *
                        * x SIGN -         se o numero for negativo insere o simbolo '-' no
                        *           string sendo montado em PAD
                        *
00007D80 00007A20' H_SIGN:        .field H_sharp
00007DA0        04                .byte 04,0h
00007DA8        00
00007DB0        53                .string "SIGN"
00007DB8        49
00007DC0        47
00007DC8        4E
00007DD0                          .even
00007DD0 00007DF0'                .field SIGN
00007DF0      0D5F  SIGN:         calla lt0
00007E00 000009A0'
00007E20      0D5F                calla cbranch
00007E30 00001600'
00007E50 00007EF0'                .field SIGN1
00007E70      0D5F                calla lit
00007E80 00001230'
00007EA0        2D                byte "-",0,0,0
00007EA8        00
00007EB0        00
00007EB8        00
00007EC0      0D5F                calla hold
```

```
      00007ED0 000076C0'
 1753 00007EF0     0960  SIGN1:        RETS
 1754                    *
 1755                    * x ># a n    termina a conversao de um numero em string ,
 1756                    *             colocando na pilha o endereco e o comprimento
 1757                    *
 1758 00007F00 00007D80' H_gtsharp:    .field H_SIGN
 1759 00007F20       02                .byte 02h,0h
      00007F28       00
 1760 00007F30       23                .string "#>"
      00007F38       3E
 1761 00007F40                         .even
 1762 00007F40 00007F60'               .field gtsharp
 1763 00007F60     0D5F  gtsharp:      calla DROP
      00007F70 00000150'
 1764 00007F90     0D5F                calla HLD
      00007FA0 00007300'
 1765 00007FC0     0D5F                calla fetch
      00007FD0 00000C10'
 1766 00007FF0     0D5F                calla PAD
      00008000 000073C0'
 1767 00008020     0D5F                calla fetch
      00008030 00000C10'
 1768 00008050     0D5F                calla over
      00008060 000002A0'
 1769 00008080     0D5F                calla minus
      00008090 00002940'
 1770 000080B0     0960                RETS
 1771                    *
 1772                    * x #S y      repete # ate y=0
 1773                    *
 1774 000080C0 00007F00' H_sharpS:      .field H_gtsharp
 1775 000080E0       02                .byte 02h,0h
      000080E8       00
 1776 000080F0       23                .string "#S"
      000080F8       53
 1777 00008100                         .even
 1778 00008100 00008120'               .field sharpS
 1779 00008120     0D5F  sharpS:       calla sharp
      00008130 00007A80'
 1780 00008150     0D5F                calla DUP
      00008160 000000A0'
 1781                    *             calla OR
 1782 00008180     0D5F                calla eq0
      00008190 00000A70'
 1783 000081B0     0D5F                calla cbranch
      000081C0 00001600'
 1784 000081E0 00008120'               .field sharpS
 1785 00008200     0960                RETS
 1786                    *
 1787                    * x (.) a     converte o numero x em string e apresenta o endereco
 1788                    *             onde foi colocado
 1789                    *
 1790 00008210 000080C0' H_rdot:        .field H_sharpS
 1791 00008230       03                .byte 03h,0h
      00008238       00
 1792 00008240       28                .string "(.)"
      00008248       2E
      00008250       29
 1793 00008260                         .even
 1794 00008260 00008280'               .field rdot
 1795 00008280     0D5F  rdot:         calla DUP
      00008290 000000A0'
 1796 000082B0     0D5F                calla ABS
      000082C0 00005480'
 1797 000082E0     0D5F                calla ltsharp
      000082F0 00007870'
 1798 00008310     0D5F                calla sharpS
      00008320 00008120'
 1799 00008340     0D5F                calla ROT
      00008350 00000350'
 1800 00008370     0D5F                calla SIGN
      00008380 00007DF0'
 1801 000083A0     0D5F                calla gtsharp
      000083B0 00007F60'
 1802 000083D0     9560                move *pstk+,a0
```

```
                        *                move @hld,a1            ; #> ja deixa
000083E0      2BA0                       sra 3,a0
000083F0      1FE0                       btst 0,a0
00008400      CA06                       jrz rdotl
00008410      1020                       addk 1,a0
00008420      0BE1                       subi 8,a1
00008430      FFF7
00008440      09C2                       movi 20h,a2
00008450      0020
00008460      8C41                       movb a2,*a1
00008470      09C2  rdotl:               movi 00,a2
00008480      0000
00008490      0BE1                       subi 8,a1
000084A0      FFF7
000084B0      8C41                       movb a2,*a1
000084C0      0B80                       andi 0ffh,a0
000084D0 FFFFFF00
000084F0      0BE1                       subi 8,a1
00008500      FFF7
00008510      8C01                       movb a0,*a1
00008520      A02B                       move a1,-*pstk
00008530      0960                       RETS
                        *
                        * x . -                pega o numero , converte em string e o envia para TOB
                        *
00008540 00008210'  H_dot:               .field h_rdot
00008560        01                       .byte 01h,0h
00008568        00
00008570        2E                       .string "."
00008580                                 .even
00008580 000085A0'                       .field dot
000085A0      0D5F  dot:                   calla rdot
000085B0 00008280'
000085D0      0D5F                       calla TYPE
000085E0 00002410'
00008600      0960                       RETS
                        *
                        * x LITERAL -    compila a funcao (LIT) e o numero x no dic.
                        *
00008610 00006FD0'  H_LITERAL            .field H_ASCII
00008630        07                       .byte 07h,01h
00008638        01
00008640        4C                       .string "LITERAL"
00008648        49
00008650        54
00008658        45
00008660        52
00008668        41
00008670        4C
00008680                                 .even
00008680 000086A0'                       .field LITERAL
000086A0      0D5F  LITERAL:             calla COMPILE
000086B0 00001C00'
000086D0      0D5F                       calla lit
000086E0 00001230'
00008700      0D5F                       calla comma
00008710 000019B0'
00008730      0960                       RETS
                        *
                        * - R> x        (R-from) recupera topo da RS no PS
                        *
00008740 00008610'  H_rfrom              .field H_literal
00008760        02                       .byte 02h,0h
00008768        00
00008770        52                       .string "R>"
00008778        3E
00008780                                 .even
00008780 000087A0'                       .field rfrom
000087A0      95E0  rfrom:               Move *SP+,a0     ;Endereco de retorno do R>
000087B0      95E1                       Move *SP+,A1     ;Topo da pilha de retorno
000087C0      A00F                       Move a0,-*SP     ;Coloca o retorno de volta na pilha
000087D0      A02B                       Move A1,-*PSTK   ;Coloca no PS o topo da pilha
000087E0      0960                       RETS
                        *
                        * x >R -        (to-r) coloca topo da PS no topo da RS
                        *
```

```
000087F0 00008740' H_rto:          .field H_rfrom
00008810       02                 .byte 02h,0h
00008818       00
00008820       3E                 .string ">R"
00008828       52
00008830                          .even
00008830 00008850'                .field rto
00008850     95E0  rto:            Move *SP+,A0    ;Salva retorno do >R
00008860     9561                 Move *PSTK+,A1 ;Tira do corpo da pilha
00008870     A02F                 Move A1,-*SP    ;Salva no RS
00008880     0160                 JuMP A0
                   *
                   * - R@ x        copia topo da pilha de retorno na de parametros
                   *
00008890 000087F0' H_Rfetch:       .field H_rto
000088B0       02                 .byte 02h,0h
000088B8       00
000088C0       52                 .string "R@"
000088C8       40
000088D0                          .even
000088D0 000088F0'                .field Rfetch
000088F0     B5E0  Rfetch:         Move  *SP(32), A0
00008900     0020
00008910     A00B                 Move A0,-*PSTK
00008920     0960                 RETS
                   *
                   * - VOCABULARY name    cria o vocabulario name . Ao se executar
                   *                      name  poe o vocabulario como contexto
                   *
00008930 00008540' H_Vocabulary:   .field H_dot
00008950       0A                 .byte 10,0h
00008958       00
00008960       56                 .string "VOCABULARY"
00008968       4F
00008970       43
00008978       41
00008980       42
00008988       55
00008990       4C
00008998       41
000089A0       52
000089A8       59
000089B0                          .even
000089B0 000089D0'                .field Vocabulary
000089D0     0D5F  VOCABULARY:     calla builds
000089E0 00003EE0'
00008A00     0D5F                 calla entry
00008A10 00002790'
00008A30     0D5F                 calla comma
00008A40 000019B0'
00008A60     0D5F                 calla does
00008A70 00004DD0'
00008A90     0D5F  dovoc:          calla rfrom
00008AA0 000087A0'
00008AC0     0D5F                 calla dup
00008AD0 000000A0'
00008AF0     0D5F                 calla CONTEXT
00008B00 000025E0'
00008B20     0D5F                 calla store
00008B30 00000CB0'
00008B50     0D5F                 calla current
00008B60 000026C0'
00008B80     0D5F                 calla store
00008B90 00000CB0'
00008BB0     0960                 RETS
                   *
                   * - SERIAL -    inicializa porta serial
                   *
00008BC0 00008930' H_SERIAL:       .int h_vocabulary
00008BE0       06                 .byte 06h,0h
00008BE8       00
00008BF0       53                 .string "SERIAL"
00008BF8       45
00008C00       52
00008C08       49
00008C10       41
```

```
      00008C18        4C
 1904 00008C20                          .even
 1905 00008C20 00008C40'                .field serial
 1906 00008C40      018A  serial:                 getst a10
 1907 00008C50      0550                setf 16,0,0
 1908 00008C60      09C0                movi 11111001b,a0        ;2sb,even,7,assyncX1
      00008C70      00F9
 1909 00008C80      0580                move a0,@wmr
      00008C90 02202000
 1910 00008CB0      09C0                movi 111010b,a0          ;intclk,2400bps
      00008CC0      003A
 1911 00008CD0      0580                move a0,@wmr
      00008CE0 02202000
 1912 00008D00      09C0                movi 0101b,a0            ;trava rts,dtr
      00008D10      0005
 1913 00008D20      0580                move a0,@wcr
      00008D30 02203000
 1914 00008D50      01AA                putst a10
 1915 00008D60      0960                rets
 1916                        *
 1917                        * - INPUT -    ativa canal de comunicacao acumulando os dados
 1918                        *             recebidos em TIB
 1919                        *
 1920 00008D70 00008BC0' h_input:          .int h_serial
 1921 00008D90        05                   .byte 05h,01h
      00008D98        01
 1922 00008DA0        69                   .string "input"
      00008DA8        6E
      00008DB0        70
      00008DB8        75
      00008DC0        74
 1923 00008DD0                             .even
 1924 00008DD0 00008DF0'                   .field input
 1925 00008DF0      05A0  input:           move @fdp ,a0
      00008E00 00001810'
 1926 00008E20      0B00                   addi 2*256*8,a0
      00008E30      1000
 1927 00008E40      4CD4                   move a0,a4
 1928 00008E50      0580                   move a0,@ftib
      00008E60 00007570'
 1929 00008E80      0580                   move a0,@finp
      00008E90 00007630'
 1930 00008EB0      0B00                   addi 256*8,a0
      00008EC0      0800
 1931 00008ED0      0580                   move a0,@fpad
      00008EE0 000073F0'
 1932 00008F00      0580                   move a0,@ftobp
      00008F10 000074B0'
 1933 00008F30      09E0                   movi true,a0
      00008F40 FFFFFFFF
 1934 00008F60      0580                   move a0,@fnendbuf
      00008F70 00009330'
 1935 00008F90      09E5                   movi 2fffffh,a5 ;time out
      00008FA0 002FFFFF
 1936 00008FC0      09C0                   movi 0100111b,a0         ;destrava rts dtr
      00008FD0      0027
 1937 00008FE0      05E0                   movb a0,@wcr
      00008FF0 02203000
 1938 00009010      07E2  inputverif:      movb @rsr,a2
      00009020 02001000
 1939 00009040      1FC2                   btst 1,a2
 1940 00009050      CB02                   jrnz    inputle
 1941 00009060      3CC5                   dsj a5,inputverif
 1942 00009070      C012                   jruc inputsai
 1943 00009080      07E1  inputle:         movb @rrhr,a1
      00009090 02000000
 1944 000090B0      0B81                   andi 07fh,a1
      000090C0 FFFFFF80
 1945 000090E0      0B41                   cmpi 0dh,a1
      000090F0      FFF2
 1946 00009100      CA09                   jrz inputsai
 1947 00009110      0B41                    cmpi 0ah,a1
      00009120      FFF5
 1948 00009130      CAED                   jrz inputverif
 1949 00009140      8C24                   movb a1,*a4
 1950 00009150      1104                   addk 8,a4
```

```
 1951 00009160      09E5                    movi 2fffffh,a5
      00009170 002FFFFF
 1952 00009190      C0E7                    jruc inputverif
 1953 000091A0      09C0   inputsai:        movi 0101b,a0
      000091B0      0005
 1954 000091C0      05E0                    movb a0,@wcr     ;trava rts dtr
      000091D0 02203000
 1955 000091F0      09C0                    movi 20h,a0
      00009200      0020
 1956 00009210      8C04                    movb a0,*a4
 1957 00009220      1104                    addk 8,a4
 1958 00009230      09C0                    movi 0,a0
      00009240      0000
 1959 00009250      8004                    move a0,*a4
 1960 00009260      0960                    rets
 1961                         *
 1962                         * - NENDBUF a  contem flag indicativo de TIB vazio (TRUE=vazio)
 1963                         *
 1964 00009270 00008D70'  H_NENDBUF:        .field H_INPUT
 1965 00009290        06                    .byte 06h,0h
      00009298        00
 1966 000092A0        4E                    .string "NENDBUF"
      000092A8        45
      000092B0        4E
      000092B8        44
      000092C0        42
      000092C8        55
      000092D0        46
 1967 000092E0                              .even
 1968 000092E0 00009300'                    .field NENDBUF
 1969 00009300      0D5F   NENDBUF:          calla DOCREATE
      00009310 000010A0'
 1970 00009330 FFFFFFFF   fNENDBUF:         .int true
 1971                         *
 1972                         * (ABORT)     Inicializa pilhas . Cold start
 1973                         *
 1974 00009350 00009270'  H_rabort:         .field H_NENDBUF
 1975 00009370        07                    .byte 07h,0h
      00009378        00
 1976 00009380        28                    .string "(ABORT)"
      00009388        41
      00009390        42
      00009398        4F
      000093A0        52
      000093A8        54
      000093B0        29
 1977 000093C0                              .even
 1978 000093C0 000093E0'                    .field rabort
 1979 000093E0      0540   rabort:                setf 32,0,0
 1980 000093F0      09EF                    movi endp + 2*stksize,SP
      00009400 0001F830'
 1981 00009420      09EB                    movi endp + 3*stksize,pstk
      00009430 00024A30'
 1982 00009450      09EC                           movi endp + 4*stksize,astk
      00009460 00029C30'
 1983 00009480      09ED                   movi endp + 5*stksize,istk
      00009490 0002EE30'
 1984                         *              movi endp + 5*stksize,sp
 1985 000094B0      0D5F                    calla serial
      000094C0 00008C40'
 1986 000094E0      C000                    JRUC INNER
      000094F0      0207
 1987                         *
 1988                         * - SP0       a       variavel que contem a origem da pilha de parametros
 1989                         *
 1990 00009500 00009350'  H_SP0:            .field H_rabort
 1991 00009520        03                    .byte 03h,0h
      00009528        00
 1992 00009530        53                    .string "SP0"
      00009538        50
      00009540        4F
 1993 00009550                              .even
 1994 00009550 00009570'                    .field SP0
 1995 00009570      0D5F   SP0:              calla DOCONSTANT
      00009580 00001170'
 1996 000095A0 00024A30'  fSP0:             .int endp+3*stksize
```

```
                                *
                                * f (ABORT)" string"   Se flag=TRUE , imprime mensagem e aborta
                                *
000095C0 00009500' H_rabortquote:     .field H_SP0
000095E0       08                  .byte 08h,0h
000095E8       00
000095F0       28                  .string "(ABORT"
000095F8       41
00009600       42
00009608       4F
00009610       52
00009618       54
00009620       22                  .byte 22h,29h
00009628       29
00009630                           .even
00009630 00009650'                 .field rabortquote
00009650     9565  rabortquote:    Move *psTK+,A5
00009660     85E0                  Move *SP,A0
00009670     A00B                  move a0,-*pstk
00009680     8401                  Move *A0,A1
00009690     1200                  addk 16,a0
000096A0     0B81                  ANDi 0FFh,a1
000096B0 FFFFFF00
000096D0     1FE1                  Btst 0,A1
000096E0     CA02                  JRZ rabortquote1
000096F0     0B01                  ADDI 1,A1
00009700     0001
00009710     2461  rabortquote1:       SLL 3,A1
00009720     4020                  ADD A1,A0
00009730     800F                  Move A0,*SP
00009740     0B45                  CMPI 0,A5
00009750     FFFF
00009760     CA04                  JREQ rabortquote2
00009770     0D5F                  calla tYPE
00009780 00002410'
000097A0     C0C3                  JRUC rabort
000097B0     9560  rabortquote2:   move *pstk+,a0
000097C0     0960                  rets
                                *
                                * ABORT" string "       Monta no dic. a funcao (ABORT)" e compila a
                                *                       seguir o string.
                                *
000097D0 00008890' H_Abortquote:      .field H_rfetch
000097F0       06                  .byte 06h,0h
000097F8       00
00009800       41                  .string "ABORT"
00009808       42
00009810       4F
00009818       52
00009820       54
00009828       22                  .byte 22h
00009830                           .even
00009830 00009850'                 .field ABORTquote
00009850     0D5F  ABORTquote:         calla COMPILE
00009860 00001C00'
00009880     0D5F                  calla rabortquote
00009890 00009650'
000098B0     0D5F                  calla dotquote
000098C0 00004C80'
000098E0     0960                  RETS
                                *
                                * - [ -                 Interrompe modo de compilacao , ativando a
                                *                       interpretacao
                                *
000098F0 000097D0' H_leftbracket  .field H_ABORTquote
00009910       01                  .byte 01h,01h
00009918       01
00009920       5B                  .string "["
00009930                           .even
00009930 00009950'                 .field leftbracket
00009950     0D5F  leftbracket:    calla fal
00009960 00001470'
00009980     0D5F                  calla mode
00009990 00003A90'
000099B0     0D5F                  calla store
000099C0 00000CB0'
```

```
2050 000099E0      0960                    RETS
2051                        *
2052                        * - SP@ a       retorna o endereco atual do topo da pilha
2053                        *
2054 000099F0 000095C0' H_retstk            .field H_rabortquote
2055 00009A10        03                    .byte 03h,0h
     00009A18        00
2056 00009A20        53                    .string "SP@"
     00009A28        50
     00009A30        40
2057 00009A40                              .even
2058 00009A40 00009A60'                    .field retstk
2059 00009A60      4D60  retstk:           Move PSTK,A0
2060 00009A70      1400                    SUBK 32,A0
2061 00009A80      A008                    Move A0,-*PSTK
2062 00009A90      0960                    RETS
2063                        *
2064                        * x y U< f     comparacao sem sinal
2065                        *
2066 00009AA0 000099F0' H_Ult               .field H_retstk
2067 00009AC0        02                    .byte 02h,0h
     00009AC8        00
2068 00009AD0        55                    .string "U<"
     00009AD8        3C
2069 00009AE0                              .even
2070 00009AE0 00009B00'                    .field Ult
2071 00009B00      5642  Ult:              clr a2
2072 00009B10      9561                    Move *PSTK+,A1
2073 00009B20      9560                    Move *PSTK+,A0
2074 00009B30      4820                    CMP A1,A0
2075 00009B40      C901                    JRHS Ult2
2076 00009B50      1422                    SUBK 1,A2
2077 00009B60      A04B  Ult2:             Move A2,-*PSTK
2078 00009B70      0960                    RETS
2079                        *
2080                        * - ?STACK -   verifica se a pilha estorou seus limites, abortando
2081                        *
2082 00009B80 00009AA0' H_CSTACK:           .field H_Ult
2083 00009BA0        06                    .byte 06h,0h
     00009BA8        00
2084 00009BB0        3F                    .string "?STACK"
     00009BB8        53
     00009BC0        54
     00009BC8        41
     00009BD0        43
     00009BD8        4B
2085 00009BE0                              .even
2086 00009BE0 00009C00'                    .field cSTACK
2087 00009C00      0D5F  cSTACK:           calla retstk
     00009C10 00009A60'
2088 00009C30      0D5F                    calla SP0
     00009C40 00009570'
2089 00009C60      0D5F                    calla SWAP
     00009C70 000001E0'
2090 00009C90      0D5F                    calla Ult
     00009CA0 00009B00'
2091 00009CC0      0D5F                    calla rabortquote
     00009CD0 00009650'
2092 00009CF0        11                    .byte 17,0
     00009CF8        00
2093 00009D00        20                    .string " Stack underflow. "
     00009D08        53
     00009D10        74
     00009D18        61
     00009D20        63
     00009D28        6B
     00009D30        20
     00009D38        75
     00009D40        6E
     00009D48        64
     00009D50        65
     00009D58        72
     00009D60        66
     00009D68        6C
     00009D70        6F
     00009D78        77
```

```
     00009D80        20
2094 00009D90      0D5F                    calla retstk
     00009DA0 00009A60'
2095 00009DC0      0D5F                    calla DIP
     00009DD0 000017E0'
2096 00009DF0      0D5F                    calla lit
     00009E00 00001230'
2097 00009E20 000000FF                     .int 0ffh
2098 00009E40      0D5F                    calla plus
     00009E50 00002890'
2099 00009E70      0D5F                    calla Ult
     00009E80 00009B00'
2100 00009EA0      0D5F                    calla rabortquote
     00009EB0 00009650'
2101 00009ED0        10                    .byte 16,0
     00009ED8        00
2102 00009EE0        20                    .string " Stack-overflow "
     00009EE8        53
     00009EF0        74
     00009EF8        61
     00009F00        63
     00009F08        6B
     00009F10        2D
     00009F18        6F
     00009F20        76
     00009F28        65
     00009F30        72
     00009F38        66
     00009F40        6C
     00009F48        6F
     00009F50        77
     00009F58        20
2103 00009F60      0960                    RETS
2104                          *
2105                          * f ?MISSING   se f=false , imprime mensagem de erro , indicando
2106                          *               a palavra deixada por DEFINED nao encontrado no
2107                          *               dicionario
2108                          *
2109 00009F70 00009B80' H_CMISSING:          .field H_CSTACK
2110 00009F90        08                    .byte 08h,0h
     00009F98        00
2111 00009FA0        3F                    .string "?MISSING"
     00009FA8        4D
     00009FB0        49
     00009FB8        53
     00009FC0        53
     00009FC8        49
     00009FD0        4E
     00009FD8        47
2112 00009FE0                              .even
2113 00009FE0 0000A000'                    .field cMISSING
2114 0000A000      0D5F  CMISSING:          calla eq0
     0000A010 00000A70'
2115 0000A030      0D5F                    calla cbranch
     0000A040 00001600'
2116 0000A060 0000A480'                    .field cMISSING2
2117 0000A080      0D5F                    calla mode
     0000A090 00003A90'
2118 0000A0B0      0D5F                    calla fetch
     0000A0C0 00000C10'
2119 0000A0E0      0D5F                    calla cbranch
     0000A0F0 00001600'
2120 0000A110 0000A310'                    .field cmissingl
2121 0000A130      0D5F                    calla current
     0000A140 000026C0'
2122 0000A160      0D5F                    calla fetch
     0000A170 00000C10'
2123 0000A190      0D5F                    calla fetch
     0000A1A0 00000C10'
2124 0000A1C0      0D5F                    calla dup
     0000A1D0 000000A0'
2125 0000A1F0      0D5F                    calla dip
     0000A200 000017E0'
2126 0000A220      0D5F                    calla store
     0000A230 00000CB0'
2127 0000A250      0D5F                    calla fetch
```

```
      0000A260 00000C10'
 2128 0000A280     0D5F                    calla current
      0000A290 000026C0'
 2129 0000A2B0     0D5F                    calla fetch
      0000A2C0 00000C10'
 2130 0000A2E0     0D5F                    calla store
      0000A2F0 00000CB0'
 2131 0000A310     0D5F  cmissing1:        calla here
      0000A320 00001720'
 2132 0000A340     0D5F                    calla lit
      0000A350 00001230'
 2133 0000A370 00000020                    .int 32
 2134 0000A390     0D5F                    calla plus
      0000A3A0 00002890'
 2135 0000A3C0     0D5F                    calla type
      0000A3D0 00002410'
 2136 0000A3F0     0D5F                    calla tru
      0000A400 000013B0'
 2137 0000A420     0D5F                    calla rabortquote
      0000A430 00009650'
 2138 0000A450       04                    .byte 04,0
      0000A458       00
 2139 0000A460       20                    .string " ?? "
      0000A468       3F
      0000A470       3F
      0000A478       20
 2140 0000A480     0960  cmissing2:        rets
 2141                    *
 2142                    * - DEFINED a f        rotina de buscas nos vocabularios
 2143                    *                       retorna endereco e flag=true se encontrou
 2144                    *                       endereco e flag=-1 se imediata
 2145                    *                       word-buffer e false se nao encontrou
 2146                    *
 2147 0000A490 00009F70' H_DEFINED:        .field H_CMISSING
 2148 0000A4B0       07                    .byte 07h,0h
      0000A4B8       00
 2149 0000A4C0       44                    .string "DEFINED"
      0000A4C8       45
      0000A4D0       46
      0000A4D8       49
      0000A4E0       4E
      0000A4E8       45
      0000A4F0       44
 2150 0000A500                             .even
 2151 0000A500 0000A520'                   .field DEFINED
 2152 0000A520     0D5F  DEFINED:          calla HERE
      0000A530 00001720'
 2153 0000A550     0D5F                    calla CONTEXT
      0000A560 000025E0'
 2154 0000A580     0D5F                    calla fetch
      0000A590 00000C10'
 2155 0000A5B0     0D5F                    calla fetch
      0000A5C0 00000C10'
 2156 0000A5E0     0D5F                    calla find
      0000A5F0 000041E0'
 2157 0000A610     0D5F                    calla DUP
      0000A620 000000A0'
 2158 0000A640     0D5F                    calla eq0
      0000A650 00000A70'
 2159 0000A670     0D5F                    calla cbranch
      0000A680 00001600'
 2160 0000A6A0 0000A9B0'                   .field DEFINED2
 2161 0000A6C0     0D5F                    calla DROP
      0000A6D0 00000150'
 2162                    *                  calla here
 2163 0000A6F0     0D5F                    calla CURRENT
      0000A700 000026C0'
 2164 0000A720     0D5F                    calla fetch
      0000A730 00000C10'
 2165 0000A750     0D5F                    calla fetch
      0000A760 00000C10'
 2166 0000A780     0D5F                    calla find
      0000A790 000041E0'
 2167 0000A7B0     0D5F                    calla DUP
      0000A7C0 000000A0'
 2168 0000A7E0     0D5F                    calla eq0
```

```
      0000A7F0 00000A7D'
 2169 0000A810     0D5F                    calla MODE
      0000A820 00003A90'
 2170 0000A840     0D5F                    calla fetch
      0000A850 00000C1D'
 2171 0000A870     0D5F                    calla AND
      0000A880 000004E0'
 2172 0000A8A0     0D5F                    calla cbranch
      0000A8B0 00001600'
 2173 0000A8D0 0000A9B0'                   .field DEFINED2
 2174 0000A8F0     0D5F                    calla DROP
      0000A900 00000150'
 2175                     *                calla here
 2176 0000A920     0D5F                    calla COMPILer
      0000A930 00002A20'
 2177 0000A950     0D5F                    calla fetch
      0000A960 00000C10'
 2178                     *                 calla fetch
 2179 0000A980     0D5F                    calla find
      0000A990 000041E0'
 2180 0000A9B0     0960  DEFINED2:         RETS
 2181                     *
 2182                     * - ' name       retorna o endereco de execucao da definicao name
 2183                     *
 2184 0000A9C0 0000A490' H_tick:            .field h_DEFINED
 2185 0000A9E0       01                    .byte 01h,0h
      0000A9E8       00
 2186 0000A9F0       27                    .string "'"
 2187 0000AA00                             .even
 2188 0000AA00 0000AA20'                   .field tick
 2189 0000AA20     0D5F  tick:             calla spc
      0000AA30 00001300'
 2190 0000AA50     0D5F                    calla token
      0000AA60 00002FA0'
 2191 0000AA80     0D5F                    calla DEFINED
      0000AA90 0000A520'
 2192 0000AAB0     0D5F                    calla CMISSING
      0000AAC0 0000A000'
 2193 0000AAE0     0960                    RETS
 2194                     *
 2195                     * - [COMPILE] name     similar a COMPILE porem para funcoes imediatas
 2196                     *
 2197 0000AAF0 000098F0' H_brackCOMPILE:.field h_leftbracket
 2198 0000AB10       09                    .byte 09h,01h
      0000AB18       01
 2199 0000AB20       5B                    .string "[COMPILE]"
      0000AB28       43
      0000AB30       4F
      0000AB38       4D
      0000AB40       50
      0000AB48       49
      0000AB50       4C
      0000AB58       45
      0000AB60       5D
 2200 0000AB70                             .even
 2201 0000AB70 0000AB90'                   .field brackCOMPILE
 2202 0000AB90     0D5F  brackCOMPILE:     calla tick
      0000ABA0 0000AA20'
 2203 0000ABC0     0D5F                    calla comma
      0000ABD0 00001980'
 2204 0000ABF0     0960                    RETS
 2205                     *
 2206                     * - ['] name     idem a "tick" porem para funcoes imediatas
 2207                     *
 2208 0000AC00 0000AAF0' H_bracktick:       .field h_brackcompile
 2209 0000AC20       03                    .byte 03h,01h
      0000AC28       01
 2210 0000AC30       5B                    .string "['"
      0000AC38       27
      0000AC40       50
 2211 0000AC50                             .even
 2212 0000AC50 0000AC70'                   .field bracktick
 2213 0000AC70     0D5F  bracktick:        calla tick
      0000AC80 0000AA20'
 2214 0000ACA0     0D5F                    calla LITERAL
      0000ACB0 000086A0'
```

```
2215 0000ACD0      0960                    RETS
2216                          *
2217                          * x 0<> f      f= true se x diferente de zero
2218                          *
2219 0000ACE0 0000A9C0' H_dif0:                  .field h_tick
2220 0000AD00       03                   .byte 03h,0h
     0000AD08       00
2221 0000AD10       30                    .string "0<>"
     0000AD18       3C
     0000AD20       3E
2222 0000AD30                             .even
2223 0000AD30 0000AD50'                   .field dif0
2224 0000AD50      5642  dif0:            clr a2
2225 0000AD60      9560                   move *pstk+,a0
2226 0000AD70      4840                   cmp a2,a0
2227 0000AD80      CA01                   jreq dif02
2228 0000AD90      03E2                   not a2
2229 0000ADA0      A04B  dif02:           move a2,-*pstk
2230 0000ADB0      0960                   rets
2231                          *
2232                          * a f INTERPRETER      rotina de controle do modo interpretativo
2233                          *
2234 0000ADC0 0000ACE0' h_interpreter: .field h_dif0
2235 0000ADE0       0B                    .byte 11,0
     0000ADE8       00
2236 0000ADF0       49                    .string "INTERPRETER"
     0000ADF8       4E
     0000AE00       54
     0000AE08       45
     0000AE10       52
     0000AE18       50
     0000AE20       52
     0000AE28       45
     0000AE30       54
     0000AE38       45
     0000AE40       52
2237 0000AE50                             .even
2238 0000AE50 0000AE70'                   .field interpreter
2239 0000AE70      0D5F  interpreter:     calla dif0
     0000AE80 0000AD50'
2240 0000AEA0      0D5F                   calla cbranch
     0000AEB0 00001600'
2241 0000AED0 0000AF70'                   .field interpret2
2242 0000AEF0      0D5F                   calla execute
     0000AF00 00000FE0'
2243 0000AF20      0D5F                   calla branch
     0000AF30 00001530'
2244 0000AF50 0000B000'                   .field interpret3
2245 0000AF70      0D5F  interpret2:      calla drop
     0000AF80 00000150'
2246 0000AFA0      0D5F                   calla number
     0000AFB0 00006C20'
2247 0000AFD0      0D5F                   calla cmissing
     0000AFE0 0000A000'
2248 0000B000      0960  interpret3:      rets
2249                          *
2250                          * a f COMPILATOR       rotina de controle do modo compilativo
2251                          *
2252 0000B010 0000ADC0' h_compilator:  .field h_interpreter
2253 0000B030       0A                    .byte 10,0
     0000B038       00
2254 0000B040       63                    .string "compilator"
     0000B048       6F
     0000B050       6D
     0000B058       70
     0000B060       69
     0000B068       6C
     0000B070       61
     0000B078       74
     0000B080       6F
     0000B088       72
2255 0000B090                             .even
2256 0000B090 0000B0B0'                   .field compilator
2257 0000B0B0      0D5F  compilator:      calla dup
     0000B0C0 000000A0'
2258 0000B0E0      0D5F                   calla dif0
```

```
      0000B0F0 0000AD50'
 2259 0000B110      0D5F                    calla cbranch
      0000B120 00001600'
 2260 0000B140 0000B390'                    .field compl
 2261 0000B160      0D5F                    calla tru
      0000B170 000013B0'
 2262 0000B190      0D5F                    calla eq
      0000B1A0 000007E0'
 2263 0000B1C0      0D5F                    calla cbranch
      0000B1D0 00001600'
 2264 0000B1F0 0000B310'                    .field comp2
 2265 0000B210      0D5F                    calla lit
      0000B220 00001230'
 2266 0000B240 00000D5F                     .int .0d5fh
 2267 0000B260      0D5F                    calla ccomma
      0000B270 00001AA0'
 2268 0000B290      0D5F                    calla comma
      0000B2A0 000019B0'
 2269 0000B2C0      0D5F                    calla branch
      0000B2D0 00001530'
 2270 0000B2F0 0000B340'                    .field comp3
 2271 0000B310      0D5F  comp2:            calla execute
      0000B320 00000FE0'
 2272 0000B340      0D5F  comp3:            calla branch
      0000B350 00001530'
 2273 0000B370 0000B4E0'                    .field comp4
 2274 0000B390      0D5F  compl:            calla drop
      0000B3A0 00000150'
 2275 0000B3C0      0D5F                    calla drop
      0000B3D0 00000150'
 2276 0000B3F0      0D5F                    calla number
      0000B400 00006C20'
 2277 0000B420      0D5F                    calla cmissing
      0000B430 0000A000'
 2278 0000B450      0D5F                    calla compile
      0000B460 00001C00'
 2279 0000B480      0D5F                    calla lit
      0000B490 00001230'
 2280 0000B4B0      0D5F                    calla comma
      0000B4C0 000019B0'
 2281 0000B4E0      0960  comp4:            rets
 2282                     *
 2283                     * - INNER -      sistema operacional basico
 2284                     *
 2285 0000B4F0 0000B010' h_inner:           .field h_compilator
 2286 0000B510        05                    .byte 05,0
      0000B518        00
 2287 0000B520        69                    .string "inner"
      0000B528        6E
      0000B530        6E
      0000B538        65
      0000B540        72
 2288 0000B550                              .even
 2289 0000B550 0000B570'                    .field inner
 2290 0000B570      0D5F  inner:            calla input
      0000B580 00008DF0'
 2291 0000B5A0      0D5F                    calla spc
      0000B5B0 00001300'
 2292 0000B5D0      0D5F                    calla token
      0000B5E0 00002FA0'
 2293 0000B600      0D5F  innerl:                   calla nendbuf
      0000B610 00009300'
 2294 0000B630      0D5F                    calla fetch
      0000B640 00000C10'
 2295 0000B660      0D5F                    calla cbranch
      0000B670 00001600'
 2296 0000B690 0000B920'                    .field inner4
 2297 0000B6B0      0D5F                    calla cstack
      0000B6C0 00009C00'
 2298 0000B6E0      0D5F                    calla defined
      0000B6F0 0000A520'
 2299 0000B710      0D5F                    calla mode
      0000B720 00003A90'
 2300 0000B740      0D5F                    calla fetch
      0000B750 00000C10'
 2301 0000B770      0D5F                    calla cbranch
```

```
      0000B780 00001600'
 2302 0000B7A0 0000B840'                .field inner2
 2303 0000B7C0      0D5F                calla compilator
      0000B7D0 0000B0B0'
 2304 0000B7F0      0D5F                calla branch
      0000B800 00001530'
 2305 0000B820 0000B870'                .field inner3
 2306 0000B840      0D5F  inner2:               calla interpreter
      0000B850 0000AE70'
 2307 0000B870      0D5F  inner3:               calla spc
      0000B880 00001300'
 2308 0000B8A0      0D5F                calla token
      0000B8B0 00002FA0'
 2309 0000B8D0      0D5F                calla branch
      0000B8E0 00001530'
 2310 0000B900 0000B600'                .field inner1
 2311 0000B920      0D5F  inner4:               calla branch
      0000B930 00001530'
 2312 0000B950 0000B570'                .field inner
 2313                                   .copy "tm-asm1.001"
A0001                   ************************************************************
A0002                   *              MINISCRIPT                                  *
A0003                   *              Stefani Forth-PDL                           *
A0004                   *              MODULO GRAFICO                              *
A0005                   *              versao 1 20/04/90                           *
A0006                   *              revisao 1 20/05/90                          *
A0007                   *              revisao 2 22/06/90                          *
A0008                   ************************************************************
A0009                   *
A0010                   *              constantes e simbolos dos registradores de I/O
A0011                   *              do TMS34010
A0012                   *
A0013 00000000                                  .data
A0014          C00000B0  control                .set 0c00000b0h
A0015          C0000130  convsp                 .set 0c0000130h
A0016          C0000150  psize                  .set 0c0000150h
A0017          C0000160  pmask                  .set 0c0000160h
A0018          C0000140  convdp                 .set 0c0000140h
A0019          C00000E0  hstadrh                .set 0c00000e0h
A0020          C00000D0  hstadrl                .set 0c00000d0h
A0021          C0000100  hstctlh                .set 0c0000100h
A0022          C00000F0  hstctll                .set 0c00000f0h
A0023          C00001F0  refcnt                 .set 0c00001f0h
A0024          C0000120  intpend                .set 0c0000120h
A0025          C00001E0  dpyadr                 .set 0c00001e0h
A0026          C0000080  dpyctl                 .set 0c0000080h
A0027          C00000A0  dpyint                 .set 0c00000a0h
A0028          C0000090  dpystrt                .set 0c0000090h
A0029          C00001B0  dpytap                 .set 0c00001b0h
A0030          C00001B0  hcount                         .set 0c00001b0h
A0031          C0000010  heblnk                 .set 0c0000010h
A0032          C0000000  hesync                 .set 0c0000000h
A0033          C0000020  hsblnk                 .set 0c0000020h
A0034          C0000030  htotal                 .set 0c0000030h
A0035          C00001D0  vcount                 .set 0c00001d0h
A0036          C0000050  veblnk                 .set 0c0000050h
A0037          C0000040  vesync                 .set 0c0000040h
A0038          C0000060  vsblnk                 .set 0c0000060h
A0039          C0000070  vtotal                 .set 0c0000070h
A0040               0010  saddr                 .set b0
A0041               0011  sptch                 .set b1
A0042               0012  daddr                 .set b2
A0043               0013  dptch                 .set b3
A0044               0014  offset                .set b4
A0045               0015  wstart                .set b5
A0046               0016  wend                  .set b6
A0047               0017  dydx                  .set b7
A0048               0018  color0                .set b8
A0049               0019  color1                .set b9
A0050               001D  pattrn                .set b13
A0051               001A  count                 .set b10
A0052               001B  inc1                  .set b11
A0053               001C  inc2                  .set b12
A0054                     *
A0055                     *
A0056 0000B970                                  .text
```

```
A0057                          *
A0058                          * - GRAPHICS -
A0059                          *              ativa vocabulario grafico
A0060                          *              Inicio do  vocabulario grafico
A0061                          *
A0062 0000B970 0000B4F0' h_graphics:                .int h_inner
A0063 0000B990        08                             .byte 08h,0h
      0000B998        00
A0064 0000B9A0        47                            .string "GRAPHICS"
      0000B9A8        52
      0000B9B0        41
      0000B9B8        50
      0000B9C0        48
      0000B9C8        49
      0000B9D0        43
      0000B9D8        53
A0065 0000B9E0                                      .even
A0066 0000B9E0 0000BA00'                            .field graphics
A0067 0000BA00      0D5F  graphics:                 calla dovoc
      0000BA10 00008A90'
A0068 0000BA30 00014D20' fgraphics:                 .int h_tshow  ;ultima palavra do voca-
A0069                          *                                   bulario
A0070                          * - GFUNCTION adr
A0071                          *                    Variavel de controle.
A0072                          *                    contem a funcao corrente a ser  usada nos
A0073                          *                    PIXBTL.
A0074                          *
A0075 0000BA50 0000B970' h_gfunction:               .int h_graphics
A0076 0000BA70        09                            .byte 09h,0h
      0000BA78        00
A0077 0000BA80        47                            .string "GFUNCTION"
      0000BA88        46
      0000BA90        55
      0000BA98        4E
      0000BAA0        43
      0000BAA8        54
      0000BAB0        49
      0000BAB8        4F
      0000BAC0        4E
A0078 0000BAD0                                      .even
A0079 0000BAD0 0000BAF0'                            .field gfunction
A0080 0000BAF0      0D5F  gfunction:                calla docreate
      0000BB00 000010A0'
A0081 0000BB20 00000000  fgfunction:                .int 0            ;  (replace)
A0082                          *
A0083                          * - VTRANSP adr
A0084                          *              Variavel de controle, contem true ou false.
A0085                          *              True (-1) transparency enable.
A0086                          *              False (0) transparency disable
A0087                          *
A0088 0000BB40 0000BA50' h_vtransp:                 .int h_gfunction
A0089 0000BB60        07                            .byte 07,0h
      0000BB68        00
A0090 0000BB70        56                            .string "VTRANSP"
      0000BB78        54
      0000BB80        52
      0000BB88        41
      0000BB90        4E
      0000BB98        53
      0000BBA0        50
A0091 0000BBB0                                      .even
A0092 0000BBB0 0000BBD0'                            .field vtransp
A0093 0000BBD0      0D5F  vtransp:                  calla docreate
      0000BBE0 000010A0'
A0094 0000BC00 00000000  fvtransp:                  .int 0            ;  (disable)
A0095                          *
A0096                          * - VORG adr
A0097                          *              Variavel de controle. contem a configuracao  corrente
A0098                          *              da posicao da origem (starting corner).
A0099                          *              Vide pag.7.7-TMS34010 User guide.
A0100                          *
A0101 0000BC20 0000BB40' h_vorg:                    .int h_vtransp
A0102 0000BC40        04                            .byte 04,0h
      0000BC48        00
A0103 0000BC50        56                            .string "VORG"
      0000BC58        4F
```

```
      0000BC60        52
      0000BC68        47
A0104 0000BC70                                  .even
A0105 0000BC70 0000BC90'                        .field vorg
A0106 0000BC90      005F  vorg:                 calla docreate
      0000BCA0 000010A0'
A0107 0000BCC0 00000000   fvorg:                .int 0
A0108                     *
A0109                     * f TRANSPARENCY -
A0110                     *              Seta o bit correspondente da transparencia na  variavel
A0111                     *              interna CONTROL e tambem o da origem PBH, PBV.
A0112                     *              True transparency
A0113                     *              False transparency
A0114                     *
A0115 0000BCE0 0000BC20' h_transparency:        .int h_vorg
A0116 0000BD00        0C                        .byte 12,0h
      0000BD08        00
A0117 0000BD10        54                        .string "TRANSPARENCY"
      0000BD18        52
      0000BD20        41
      0000BD28        4E
      0000BD30        53
      0000BD38        50
      0000BD40        41
      0000BD48        52
      0000BD50        45
      0000BD58        4E
      0000BD60        43
      0000BD68        59
A0118 0000BD70                                  .even
A0119 0000BD70 0000BD90'                        .field transparency
A0120 0000BD90      9560  transparency:         move *pstk+,a0
A0121 0000BDA0      0580                        move a0,@fvtransp
      0000BDB0 0000BC00'
A0122 0000BDD0      0B80                        andi 100000b, a0
      0000BDE0 FFFFFFDF
A0123 0000BE00      05A1                        move @fvorg, a1
      0000BE10 0000BCC0'
A0124 0000BE30      0B81                        andi 11b,a1
      0000BE40 FFFFFFFC
A0125 0000BE60      2501                        sll 8,a1
A0126 0000BE70      5420                        or a1,a0
A0127 0000BE80      018A                        getst a10
A0128 0000BE90      0550                        setf 16,0,0
A0129 0000BEA0      05A2                        move @control,a2
      0000BEB0 C00000B0
A0130 0000BED0      0B82                       andi 11111100110111111b,a2
      0000BEE0 FFFF0320
A0131 0000BF00      5402                        or a0,a2
A0132 0000BF10      0582                        move a2,@control
      0000BF20 C00000B0
A0133 0000BF40      01AA                        putst a10
A0134 0000BF50      0960                        rets
A0135                     *
A0136                     * x LOADFUNCTION -
A0137                     *              Seta o campo PPOpt no register  CONTROL estabelecendo
A0138                     *              operacao desejada para os proximos PIXBTL,LINE FILL.
A0139                     *
A0140 0000BF60 0000BCE0' h_loadfunction:        .int h_transparency
A0141 0000BF80        0C                        .byte 12,0h
      0000BF88        00
A0142 0000BF90        4C                        .string "LOADFUNCTION"
      0000BF98        4F
      0000BFA0        41
      0000BFA8        44
      0000BFB0        46
      0000BFB8        55
      0000BFC0        4E
      0000BFC8        43
      0000BFD0        54
      0000BFD8        49
      0000BFE0        4F
      0000BFE8        4E
A0143 0000BFF0                                  .even
A0144 0000BFF0 0000C010'                        .field loadfunction
A0145 0000C010      9560  loadfunction:         move *pstk+, a0
```

```
A0146 0000C020      0580                          move a0,@fgfunction
      0000C030 0000BB20'
A0147 0000C050      0B80                          andi 011111b, a0
      0000C060 FFFFFFED
A0148 0000C080      2540                          sll 10,a0
A0149 0000C090      018A                          getst a10
A0150 0000C0A0      0550                          setf 16,0,0
A0151 0000C0B0      05A2                          move @control, a2
      0000C0C0 C00000B0
A0152 0000C0E0      0B82                         andi 1000001111111111b,a2
      0000C0F0 FFFF7C00
A0153 0000C110      5402                          or a0,a2
A0154 0000C120      0582                          move a2,@control
      0000C130 C00000B0
A0155 0000C150      01AA                          putst a10
A0156 0000C160      0960                          rets
A0157                       *
A0158                       * - REPLACE -
A0159                       *      Primeira funcao grafica.
A0160                       *
A0161 0000C170 0000BF60' h_replace:               .int h_loadfunction
A0162 0000C190        07                          .byte 07,0h
      0000C198        00
A0163 0000C1A0        52                          .string "REPLACE"
      0000C1A8        45
      0000C1B0        50
      0000C1B8        4C
      0000C1C0        41
      0000C1C8        43
      0000C1D0        45
A0164 0000C1E0                                    .even
A0165 0J00C1E0 0000C200'                          .int replace
A0166 0000C200      0D5F  replace:                calla lit
      0000C210 00001230'
A0167 0000C230 00000000                           .int 0
A0168 0000C250      0D5F                          calla loadfunction
      0000C260 0000C010'
A0169 0000C280      0960                          rets
A0170                       *
A0171                       * - GAND -            ppop 00001
A0172                       *
A0173 0000C290 0000C170' h_gand:                  .int h_replace
A0174 0000C2B0        04                          .byte 04,0h
      0000C2B8        00
A0175 0000C2C0        47                          .string "GAND"
      0000C2C8        41
      0000C2D0        4E
      0000C2D8        44
A0176 0000C2E0                                    .even
A0177 0000C2E0 0000C300'                          .int gand
A0178 0000C300      0D5F  gand:                   calla lit
      0000C310 00001230'
A0179 0000C330 00000001                           .int 00001b
A0180 0000C350      0D5F                          calla loadfunction
      0000C360 0000C010'
A0181 0000C380      0960                          rets
A0182                       *
A0183                       * - GANDNOT -         ppop 00010
A0184                       *
A0185 0000C390 0000C290' h_gandnot:               .int h_gand
A0186 0000C3B0        07                          .byte 07,0h
      0000C3B8        00
A0187 0000C3C0        47                          .string "GANDNOT"
      0000C3C8        41
      0000C3D0        4E
      0000C3D8        44
      0000C3E0        4E
      0000C3E8        4F
      0000C3F0        54
A0188 0000C400                                    .even
A0189 0000C400 0000C420'                          .int gandnot
A0190 0000C420      0D5F  gandnot:                calla lit
      0000C430 00001230'
A0191 0000C450 00000002                           .int 00010b
A0192 0000C470      0D5F                          calla loadfunction
      0000C480 0000C010'
```

```
A0193 0000C4A0      0960                                  rets
A0194                          *
A0195                          * - BLACK -             ppop 00011
A0196                          *
A0197 0000C4B0 0000C390' h_black:                        .int h_gandnot
A0198 0000C4D0        05                                 .byte 05,0h
      0000C4D8        00
A0199 0000C4E0        42                                 .string "BLACK"
      0000C4E8        4C
      0000C4F0        41
      0000C4F8        43
      0000C500        4B
A0200 0000C510                                           .even
A0201 0000C510 0000C530'                                 .int black
A0202 0000C530      0D5F  black:                         calla lit
      0000C540 00001230'
A0203 0000C560 00000003                                  .int 00011b
A0204 0000C580      0D5F                                 calla loadfunction
      0000C590 0000C010'
A0205 0000C5B0      0960                                 rets
A0206                          *
A0207                          * - GORNOT -            ppop 00100
A0208                          *
A0209 0000C5C0 0000C4B0' h_gornot                        .int h_black
A0210 0000C5E0        06                                 .byte 06,0h
      0000C5E8        00
A0211 0000C5F0        47                                 .string "GORNOT"
      0000C5F8        4F
      0000C600        52
      0000C608        4E
      0000C610        4F
      0000C618        54
A0212 0000C620                                           .even
A0213 0000C620 0000C640'                                 .int gornot
A0214 0000C640      0D5F  gornot:                        calla lit
      0000C650 00001230'
A0215 0000C670 00000004                                  .int 00100b
A0216 0000C690      0D5F                                 calla loadfunction
      0000C6A0 0000C010'
A0217 0000C6C0      0960                                 rets
A0218                          *
A0219                          * - GXNOR -             ppop 00101
A0220                          *
A0221 0000C6D0 0000C5C0' h_gxnor:                        .int h_gornot
A0222 0000C6F0        05                                 .byte 05,0h
      0000C6F8        00
A0223 0000C700        47                                 .string "GXNOR"
      0000C708        58
      0000C710        4E
      0000C718        4F
      0000C720        52
A0224 0000C730                                           .even
A0225 0000C730 0000C750'                                 .int gxnor
A0226 0000C750      0D5F  gxnor:                         calla lit
      0000C760 00001230'
A0227 0000C780 00000005                                  .int 00101b
A0228 0000C7A0      0D5F                                 calla loadfunction
      0000C7B0 0000C010'
A0229 0000C7D0      0960                                 rets
A0230                          *
A0231                          * - GNOTD -             ppop 00110
A0232                          *
A0233 0000C7E0 0000C6D0' h_gnotd:                        .int h_gxnor
A0234 0000C800        05                                 .byte 05,0h
      0000C808        00
A0235 0000C810        47                                 .string "GNOTD"
      0000C818        4E
      0000C820        4F
      0000C828        54
      0000C830        44
A0236 0000C840                                           .even
A0237 0000C840 0000C860'                                 .int gnotd
A0238 0000C860      0D5F  gnotd:                         calla lit
      0000C870 00001230'
A0239 0000C890 00000006                                  .int 00110b
A0240 0000C8B0      0D5F                                 calla loadfunction
```

```
      0000C8C0 0000C010'
A0241 0000C8E0     0960                          rets
A0242                      *
A0243                      * - GNOR -          ppop 00111
A0244                      *
A0245 0000C8F0 0000C7E0' h_gnor:                 .int h_gnotd
A0246 0000C910       04                          .byte 04,0h
      0000C918       00
A0247 0000C920       47                          .string "GNOR"
      0000C928       4E
      0000C930       4F
      0000C938       52
A0248 0000C940                                   .even
A0249 0000C940 0000C960'                         .int gnor
A0250 0000C960     0D5F  gnor:                   calla lit
      0000C970 00001230'
A0251 0000C990 00000007                          .int 00111b
A0252 0000C980     0D5F                          calla loadfunction
      0000C9C0 0000C010'
A0253 0000C9E0     0960                          rets
A0254                      *
A0255                      * - GOR        -          ppop 01000
A0256                      *
A0257 0000C9F0 0000C8F0' h_gor:                  .int h_gnor
A0258 0000CA10       03                          .byte 03,0h
      0000CA18       00
A0259 0000CA20       47                          .string "GOR"
      0000CA28       4F
      0000CA30       52
A0260 0000CA40                                   .even
A0261 0000CA40 0000CA60'                         .int gor
A0262 0000CA60     0D5F  gor:                    calla lit
      0000CA70 00001230'
A0263 0000CA90 00000008                          .int 01000b
A0264 0000CAB0     0D5F                          calla loadfunction
      0000CAC0 0000C010'
A0265 0000CAE0     0960                          rets
A0266                      *
A0267                      * - GNOP -          ppop 01001
A0268                      *
A0269 0000CAF0 0000C9F0' h_nop:                  .int h_gor
A0270 0000CB10       03                          .byte 03,0h
      0000CB18       00
A0271 0000CB20       4E                          .string "NOP"
      0000CB28       4F
      0000CB30       50
A0272 0000CB40                                   .even
A0273 0000CB40 0000CB60'                         .int nop
A0274 0000CB60     0D5F  nop:                    calla lit
      0000CB70 00001230'
A0275 0000CB90 00000009                          .int 01001b
A0276 0000CBB0     0D5F                          calla loadfunction
      0000CBC0 0000C010'
A0277 0000CBE0     0960                          rets
A0278                      *
A0279                      * - GXOR -          ppop 01010
A0280                      *
A0281 0000CBF0 0000CAF0' h_gxor:                 .int h_nop
A0282 0000CC10       04                          .byte 04,0h
      0000CC18       00
A0283 0000CC20       47                          .string "GXOR"
      0000CC28       58
      0000CC30       4F
      0000CC38       52
A0284 0000CC40                                   .even
A0285 0000CC40 0000CC60'                         .int gxor
A0286 0000CC60     0D5F  gxor:                   calla lit
      0000CC70 00001230'
A0287 0000CC90 0000000A                          .int 01010b
A0288 0000CCB0     0D5F                          calla loadfunction
      0000CCC0 0000C010'
A0289 0000CCE0     0960                          rets
A0290                      *
A0291                      * - GNOTAND -       ppop 01011
A0292                      *
A0293 0000CCF0 0000CBF0' h_gnotand:              .int h_gxor
```

```
A0294 0000CD10        07                          .byte 07,0h
      0000CD18        00
A0295 0000CD20        47                          .string "GNOTAND"
      0000CD28        4E
      0000CD30        4F
      0000CD38        54
      0000CD40        41
      0000CD48        4E
      0000CD50        44
A0296 0000CD60                                    .even
A0297 0000CD60 0000CD80'                          .int gnotand
A0298 0000CD80      0D5F  gnotand:                calla lit
      0000CD90 00001230'
A0299 0000CDB0 0000000B                           .int 01011b
A0300 0000CDD0      0D5F                          calla loadfunction
      0000CDE0 0000C010'
A0301 0000CE00      0960                          rets
A0302                     *
A0303                     * - WHITE -           ppop 01100
A0304                     *
A0305 0000CE10 0000CCFD' h_white:                 .int h_gnotand
A0306 0000CE30        05                          .byte 05,0h
      0000CE38        00
A0307 0000CE40        57                          .string "WHITE"
      0000CE48        48
      0000CE50        49
      0000CE58        54
      0000CE60        45
A0308 0000CE70                                    .even
A0309 0000CE70 0000CE90'                          .int white
A0310 0000CE90      0D5F  white:                  calla lit
      0000CEA0 00001230'
A0311 0000CEC0 0000000C                           .int 01100b
A0312 0000CEE0      0D5F                          calla loadfunction
      0000CEF0 0000C010'
A0313 0000CF10      0960                          rets
A0314                     *
A0315                     * - GNOTOR -          ppop 01101
A0316                     *
A0317 0000CF20 0000CE10' h_gnotor:                .int h_white
A0318 0000CF40        06                          .byte 06,0h
      0000CF48        00
A0319 0000CF50        47                          .string "GNOTOR"
      0000CF58        4E
      0000CF60        4F
      0000CF68        54
      0000CF70        4F
      0000CF78        52
A0320 0000CF80                                    .even
A0321 0000CF80 0000CFA0'                          .int gnotor
A0322 0000CFA0      0D5F  gnotor:                 calla lit
      0000CFB0 00001230'
A0323 0000CFD0 0000000D                           .int 01101b
A0324 0000CFF0      0D5F                          calla loadfunction
      0000D000 0000C010'
A0325 0000D020      0960                          rets
A0326                     *
A0327                     * - GNAND -           ppop 01110
A0328                     *
A0329 0000D030 0000CF20' h_gnand:                 .int h_gnotor
A0330 0000D050        05                          .byte 05,0h
      0000D058        00
A0331 0000D060        47                          .string "GNAND"
      0000D068        4E
      0000D070        41
      0000D078        4E
      0000D080        44
A0332 0000D090                                    .even
A0333 0000D090 0000D0B0'                          .int gnand
A0334 0000D0B0      0D5F  gnand:                  calla lit
      0000D0C0 00001230'
A0335 0000D0E0 0000000E                           .int 01110b
A0336 0000D100      0D5F                          calla loadfunction
      0000D110 0000C010'
A0337 0000D130      0960                          rets
A0338                     *
```

```
A0339                          * - GNOTS -          ppop 01111
A0340                          *
A0341 0000D140 0000D030' h_gnots:                     .int h_gnand
A0342 0000D160       05                               .byte 05,0h
      0000D168       00
A0343 0000D170       47                               .string "GNOTS"
      0000D178       4E
      0000D180       4F
      0000D188       54
      0000D190       53
A0344 0000D1A0                                        .even
A0345 0000D1A0 0000D1C0'                              .int gnots
A0346 0000D1C0     0D5F  gnots:                      calla lit
      0000D1D0 00001230'
A0347 0000D1F0 0000000F                               .int 01111b
A0348 0000D210     0D5F                               calla loadfunction
      0000D220 0000C010'
A0349 0000D240     0960                               rets
A0350                          *
A0351                          * - GPLUS -          ppop 10000
A0352                          *
A0353 0000D250 0000D140' h_gplus:                     .int h_gnots
A0354 0000D270       05                               .byte 05,0h
      0000D278       00
A0355 0000D280       47                               .string "GPLUS"
      0000D288       50
      0000D290       4C
      0000D298       55
      0000D2A0       53
A0356 0000D2B0                                        .even
A0357 0000D2B0 0000D2D0'                              .int gplus
A0358 0000D2D0     0D5F  gplus:                      calla lit
      0000D2E0 00001230'
A0359 0000D300 00000010                               .int 10000b
A0360 0000D320     0D5F                               calla loadfunction
      0000D330 0000C010'
A0361 0000D350     0960                               rets
A0362                          *
A0363                          * - GADDSAT -        ppop 10001
A0364                          *
A0365 0000D360 0000D250' h_gaddsat:                   .int h_gplus
A0366 0000D380       07                               .byte 07,0h
      0000D388       00
A0367 0000D390       47                               .string "GADDSAT"
      0000D398       41
      0000D3A0       44
      0000D3A8       44
      0000D3B0       53
      0000D3B8       41
      0000D3C0       54
A0368 0000D3D0                                        .even
A0369 0000D3D0 0000D3F0'                              .int gaddsat
A0370 0000D3F0     0D5F  gaddsat:                    calla lit
      0000D400 00001230'
A0371 0000D420 00000011                               .int 10001b
A0372 0000D440     0D5F                               calla loadfunction
      0000D450 0000C010'
A0373 0000D470     0960                               rets
A0374                          *
A0375                          * - GMINUS -         ppop 10010
A0376                          *
A0377 0000D480 0000D360' h_gminus:                    .int h_gaddsat
A0378 0000D4A0       06                               .byte 06,0h
      0000D4A8       00
A0379 0000D4B0       47                               .string "GMINUS"
      0000D4B8       4D
      0000D4C0       49
      0000D4C8       4E
      0000D4D0       55
      0000D4D8       53
A0380 0000D4E0                                        .even
A0381 0000D4E0 0000D500'                              .int gminus
A0382 0000D500     0D5F  gminus:                     calla lit
      0000D510 00001230'
A0383 0000D530 00000012                               .int 10010b
A0384 0000D550     0D5F                               calla loadfunction
```

```
      0000D560 0000C010'
A0385 0000D580     0960                          rets
A0386                      *
A0387                      * - GSUBSAT -       ppop 10010
A0388                      *
A0389 0000D590 0000D480' h_gsubsat:              .int h_gminus
A0390 0000D5B0       07                          .byte 07,0h
      0000D5B8       00
A0391 0000D5C0       47                          .string "GSUBSAT"
      0000D5C8       53
      0000D5D0       55
      0000D5D8       42
      0000D5E0       53
      0000D5E8       41
      0000D5F0       54
A0392 0000D600                                   .even
A0393 0000D600 0000D620'                         .int gsubsat
A0394 0000D620     0D5F  gsubsat:                calla lit
      0000D630 00001230'
A0395 0000D650 00000013                          .int 10011b
A0396 0000D670     0D5F                          calla loadfunction
      0000D680 0000C010'
A0397 0000D6A0     0960                          rets
A0398                      *
A0399                      * - GMAX -          ppop 10011
A0400                      *
A0401 0000D6B0 0000D590' h_gmax:                 .int h_gsubsat
A0402 0000D6D0       05                          .byte 05,0h
      0000D6D8       00
A0403 0000D6E0       47                          .string "GMAX"
      0000D6E8       4D
      0000D6F0       41
      0000D6F8       58
A0404 0000D700                                   .even
A0405 0000D700 0000D720'                         .int gmax
A0406 0000D720     0D5F  gmax:                   calla lit
      0000D730 00001230'
A0407 0000D750 00000014                          .int 10100b
A0408 0000D770     0D5F                          calla loadfunction
      0000D780 0000C010'
A0409 0000D7A0     0960                          rets
A0410                      *
A0411                      * - GMIN -          ppop 10011
A0412                      *
A0413 0000D7B0 0000D6B0' h_gmin:                 .int h_gmax
A0414 0000D7D0       04                          .byte 04,0h
      0000D7D8       00
A0415 0000D7E0       47                          .string "GMIN"
      0000D7E8       4D
      0000D7F0       49
      0000D7F8       4E
A0416 0000D800                                   .even
A0417 0000D800 0000D820'                         .int gmin
A0418 0000D820     0D5F  gmin:                   calla lit
      0000D830 00001230'
A0419 0000D850 00000015                          .int 10101b
A0420 0000D870     0D5F                          calla loadfunction
      0000D880 0000C010'
A0421 0000D8A0     0960                          rets
A0422                      *
A0423                      * pitch dy dx adr  DOSTENCIL name
A0424                      *
A0425                      *    Rotina inicializadora dos stencils era no dicionario um regis-
A0426                      *    tro da forma:
A0427                      *
A0428                      *   +---------------------+
A0429                      *   |        nome         | :header igual as outras funcoes ;com
A0430                      *   |---------------------|  um calla RDOSTENCIL
A0431                      *   | endereco  inicial   | :endereco na memoria grafica
A0432                      *   |---------------------|
A0433                      *   |   DY      |   DX    | :dimensao DY x DX pontos
A0434                      *   |---------------------|
A0435                      *   |   CPX     |   CPY   | :posicao do pointer dentro do stencil
A0436                      *   |---------------------|
A0437                      *   |        DPTH         | :pitch
A0438                      *   +---------------------+
```

```
A0439                          *
A0440                          *       Pitch DY DX endereco DOSTENCIL nome
A0441                          *
A0442 0000D8B0 0000D7B0' h_dostencil:          .int h_gmin
A0443 0000D8D0       09                        .byte 09,0h
      0000D8D8       00
A0444 0000D8E0       44                        .string "DOSTENCIL"
      0000D8E8       4F
      0000D8F0       53
      0000D8F8       54
      0000D900       45
      0000D908       4E
      0000D910       43
      0000D918       49
      0000D920       4C
A0445 0000D930                                 .even
A0446 0000D930 0000D950'                       .field dostencil
A0447 0000D950     0D5F  dostencil:            calla builds      ;(deixa here na pilha)
      0000D960 00003EE0'
A0448 0000D980     9560                        move *pstk+, a0   ;contem here/does
A0449 0000D990     9561                        move *pstk+,a1  ;contem endereco inicial
A0450 0000D9A0     9562                        move *pstk+,a2    ;contem dx
A0451 0000D9B0     9563                        move *pstk+,a3    ;contem dy
A0452 0000D9C0     9564                        move *pstk+,a4    ;contem pitch
A0453 0000D9D0     05A5                        move @fdp,a5      ;contem pointer no di-
      0000D9E0 00001810'
A0454                          *                                  cionario
A0455 0000DA00     A00B                        move a0,-*pstk    ;contem devolve here/does
A0456 0000DA10     9025                        move a1,*a5+      ;salva endereco inicial
A0457 0000DA20     0B82                        andi 0ffffh,a2    ;limpa x
      0000DA30 FFFF0000
A0458 0000DA50     0B83                        andi 0ffffh,a3    ;limpa y
      0000DA60 FFFF0000
A0459 0000DA80     2603                        sll 16,a3         ;move y
A0460 0000DA90     EE62                        movy a3,a2
A0461 0000DAA0     9045                        move a2, *a5+     ;poe no dicionario
A0462 0000DAB0     5600                        clr a0
A0463 0000DAC0     9005                        move a0, *a5+     ;poe condicao inicial em
A0464                          *                                 cpx  cpy
A0465 0000DAD0     9085                        move a4,*a5+      ;poe dptch no dia
A0466 0000DAE0     0585                        move a5,@fdp      ;atualiza dip
      0000DAF0 00001810'
A0467 0000DB10     0D5F                        calla does        ;quando executado deixa o
      0000DB20 000040D0'
A0468 0000DB40     95E0  rdostencil:           move *sp+,a0      ;endereco do rdostencil
A0469 0000DB50     A00B                        move a0,-*pstk
A0470 0000DB60     0960                        rets
A0471                          *
A0472                          * - GORIGIN adr
A0473                          *       Pointer dentro do frame buffer, ou memoria do video indican
A0474                          *       do a posicao vaga para o stencial sendo criado
A0475                          *
A0476 0000DB70 0000D8B0' h_gorigin:                .int h_dostencil
A0477 0000DB90       07                            .byte 07,0h
      0000DB98       00
A0478 0000DBA0       47                            .string "GORIGIN"
      0000DBA8       4F
      0000DBB0       52
      0000DBB8       49
      0000DBC0       47
      0000DBC8       49
      0000DBD0       4E
A0479 0000DBE0                                     .even
A0480 0000DBE0 0000DC00'                           .field gorigin
A0481 0000DC00     0D5F  gorigin:                  calla docreate
      0000DC10 000010A0'
A0482 0000DC30 00000000  fgorigin:                 .int 0
A0483                          *
A0484                          *
A0485                          * dy dx STENCIL name
A0486                          *       Cria stencil padrao seguindo a posicao da memoria disponivel
A0487                          *       indicado por gorigin, com as dimensoes dy dx.
A0488                          *
A0489                          *
A0490 0000DC50 0000DB70' h_stencil:                .int h_gorigin
A0491 0000DC70       07                            .byte 07,0h
```

```
      0000DC78        00
A0492 0000DC80        53                          .string "STENCIL"
      0000DC88        54
      0000DC90        45
      0000DC98        4E
      0000DCA0        43
      0000DCA8        49
      0000DCB0        4C
A0493 0000DCC0                                    .even
A0494 0000DCC0 0000DCE0'                          .field stencil
A0495 0000DCE0      9561  stencil:                move *pstk+,a1   ;  dx
A0496 0000DCF0      9560                          move *pstk+,a0   ;  dy
A0497 0000DD00      A00B                          move a0,-*pstk   ;pitch
A0498 0000DD10      A00B                          move a0,-*pstk   ;dy
A0499 0000DD20      A02B                          move a1,-*pstk   ;dx
A0500 0000DD30      05A2                          move @fgorigin,a2
      0000DD40 0000DC30'
A0501 0000DD60      A04B                          move a2,-*pstk       ;address
A0502 0000DD70      A00B                          move a0,-*pstk
A0503 0000DD80      A02B                          move a1,-*pstk
A0504 0000DD90      0D5F                          calla times
      0000DDA0 00004E80'
A0505 0000DDC0      9560                          move *pstk+,a0
A0506 0000DDD0      05A2                          move @fgorigin,a2
      0000DDE0 0000DC30'
A0507 0000DE00      4002                          add a0,a2
A0508 0000DE10      0582                          move a2,@fgorigin
      0000DE20 0000DC30'
A0509 0000DE40      0D5F                          calla dostencil
      0000DE50 0000D950'
A0510 0000DE70      0960                          rets
A0511                       *
A0512                       * adr adr COMPOSE -
A0513                       *       Source Dest COMPOSE
A0514                       *       Acao de composicao.
A0515                       *       PIXBTL L,L
A0516                       *
A0517 0000DE80 0000DC50' h_compose:               .int h_stencil
A0518 0000DEA0        07                          .byte 07,0h
      0000DEA8        00
A0519 0000DEB0        43                          .string "COMPOSE"
      0000DEB8        4F
      0000DEC0        4D
      0000DEC8        50
      0000DED0        4F
      0000DED8        53
      0000DEE0        45
A0520 0000DEF0                                    .even
A0521 0000DEF0 0000DF10'                          .field compose
A0522 0000DF10      9560  compose:                move *pstk+,a0
A0523 0000DF20      9401                          move *a0+,a1         ;endereco dest
A0524 0000DF30      9402                          move *a0+,a2         ;dx  dy   dest
A0525 0000DF40      9403                          move *a0+,a3         ;cpy, cpx
A0526 0000DF50      9403                          move *a0+,a3         ;dpitch
A0527 0000DF60      4E63                          move a3,b3          ;(dptch)
A0528 0000DF70      4E47                          move a2,b7          ;(dydx)
A0529 0000DF80      4E22                          move a1,b2          ;(daddr)
A0530 0000DF90      9560                          move *pstk+,a0       ;endereco source
A0531 0000DFA0      9401                          move *a0+,a1
A0532 0000DFB0      9402                          move *a0+,a2         ;s  dx  dy
A0533 0000DFC0      9403                          move *a0+,a3
A0534 0000DFD0      9403                          move *a0+,a3         ;spitch
A0535 0000DFE0      4E20                          move a1,b0           ;saddr
A0536 0000DFF0      4E61                          move a3,b1           ;sptch
A0537 0000E000      0F00                          pixblt L,L
A0538 0000E010      0960                          rets
A0539                       *
A0540                       * adr adr PASTE -
A0541                       *       Source Dest PASTE
A0542                       *       (PIXBTL  L,XY) O source e colocado em D(CPX,CPY).
A0543                       *       Apos  a operacao, D(CPX,CPY) e atualizado da forma:
A0544                       *       s(CPX,CPY+ D(CPX,CPY) = D(CPX,CPY)
A0545                       *
A0546 0000E020 0000DE80' h_paste:                 .int h_compose
A0547 0000E040        05                          .byte 05,0
      0000E048        00
```

```
A0548 0000E050        50                          .string "PASTE"
      0000E058        41
      0000E060        53
      0000E068        54
      0000E070        45
A0549 0000E080                                    .even
A0550 0000E080 0000E0A0'                          .field paste
A0551 0000E0A0      9560  paste:                  move *pstk+,a0
A0552 0000E0B0      9401                          move *a0+,a1     ;endereco dest
A0553 0000E0C0      9402                          move *a0+,a2     ;dxdy dest
A0554 0000E0D0      4C08                          move a0,a8       ;endereco (px,py)
A0555 0000E0E0      9403                          move *a0+,a3     ;cpx,cpy
A0556 0000E0F0      4C67                          move a3,a7       ;cpx,cpy
A0557 0000E100      9404                          move *a0+,a4     ;pitch
A0558 0000E110      4E24                          move a1,b4       ;offset
A0559 0000E120      4E62                          move a3,b2       ;daddr - (cpx,cpy)
A0560 0000E130      4E83                          move a4,b3       ;dptch
A0561 0000E140      018A                          getst a10        ;
A0562 0000E150      0550                          setf 16,0,0
A0563 0000E160      6A89                          lmo a4,a9
A0564 0000E170      0589                          move a9, @convdp
      0000E180 C0000140
A0565 0000E1A0      01AA                          putst a10
A0566 0000E1B0      9560                          move *pstk+,a0   ;
A0567 0000E1C0      9401                          move *a0+, a1    ;endereco source
A0568 0000E1D0      9402                          move *a0+,a2     ;dydx
A0569 0000E1E0      9403                          move *a0+,a3     ;cpx,cpy
A0570 0000E1F0      9404                          move *a0+,a4     ;pitch
A0571 0000E200      4E20                          move a1,b0       ;saddr
A0572 0000E210      4E81                          move a4,b1       ;sptch
A0573 0000E220      4E47                          move a2,b7       ;dydx
A0574 0000E230      018A                          getst a10
A0575 0000E240      0550                          setf 16,0,0      ;prepare convsp
A0576 0000E250      6A89                          lmo a4,a9
A0577 0000E260      0589                          move a9,@convsp
      0000E270 C0000130
A0578 0000E290      01AA                          putst a10
A0579 0000E2A0      E067                          addxy  a3,a7     ;soma s(cpx,cpy)
A0580                       *                                        em d(cpx,cpy)
A0581 0000E2B0      80E8                          move a7,*a8      ;salva em d(cpx,cpy)
A0582 0000E2C0      0F20                          pixblt l,xy
A0583 0000E2D0      0960                          rets
A0584                       *
A0585                       * adr adr DETACH -
A0586                       *      Source Dest DETACH
A0587                       *       (PIXBTL XY, L)
A0588                       *      e retirado  (copiado  do source um retangulo D(dy,dx) apontado
A0589                       *      por: s(cpx,cpy) e colocado em daddr
A0590                       *
A0591 0000E2E0 0000E020' h_detach:                .int h_paste
A0592 0000E300        06                          .byte 06,0
      0000E308        00
A0593 0000E310        44                          .string "DETACH"
      0000E318        45
      0000E320        54
      0000E328        41
      0000E330        43
      0000E338        48
A0594 0000E340                                    .even
A0595 0000E340 0000E360'                          .field detach
A0596 0000E360      9560  detach:                 move *pstk+,a0
A0597 0000E370      9401                          move *a0+,a1     ;daddr
A0598 0000E380      9402                          move *a0+,a2     ;d(dy,dx)
A0599 0000E390      9403                          move *a0+,a3     ;d(cpx,cpy)
A0600 0000E3A0      9404                          move *a0+,a4     ;dptch
A0601 0000E3B0      4E22                          move a1,b2       ;daddr
A0602 0000E3C0      4E83                          move a4,b3       ;dptch
A0603 0000E3D0      4E47                          move a2,b7       ;dydx
A0604 0000E3E0      018A                          getst a10
A0605 0000E3F0      0550                          setf 16,0,0
A0606 0000E400      6A89                          lmo a4,a9
A0607 0000E410      0589                          move a9, @convdp
      0000E420 C0000140
A0608 0000E440      01AA                          putst a10
A0609 0000E450      9560                          move *pstk+,a0
A0610 0000E460      9401                          move *a0+,a1     ;saddr
```

```
A0611 0000E470      9402                          move *a0+,a2    ;s(dy,dx)
A0612 0000E480      9403                          move *a0+,a3    ;s(cpx,cpy)
A0613 0000E490      9404                          move *a0+,a4    ;sptch
A0614 0000E4A0      4E24                          move a1,b4      ;offset
A0615 0000E4B0      4E81                          move a4,b1      ;sptch
A0616 0000E4C0      4E60                          move a3,b0      ;saddr
A0617 0000E4D0      018A                          getst a10
A0618 0000E4E0      0550                          setf 16,0,0
A0619 0000E4F0      6A89                          lmo a4,a9       ;prepara convsp
A0620 0000E500      0589                          move a9,@convsp
      0000E510 C0000130
A0621 0000E530      01AA                          putst a10
A0622 0000E540      0F40                          pixblt xy,1
A0623 0000E550      0960                          rets
A0624                       *
A0625                       * adr y x y x REPASTE -
A0626                       *      Source NY  NX  RDY  RDX REPASTE
A0627                       *      (PIXBTL  xy,xy)
A0628                       *      Copia o stencial de tamanho RDX, RDY apontado
A0629                       *      por S(CPX, CPY) em S(NX, NY).
A0630                       *      Cuidado em nao sobrepor , caso  contrario e necessario
A0631                       *      alterar os parametros PBH, PBV, correspondentemente.
A0632                       *
A0633 0000E560 0000E2E0' h_repaste:                 .int h_detach
A0634 0000E580        07                            .byte 07,0
      0000E588        00
A0635 0000E590        52                            .string "REPASTE"
      0000E598        45
      0000E5A0        50
      0000E5A8        41
      0000E5B0        53
      0000E5B8        54
      0000E5C0        45
A0636 0000E5D0                                      .even
A0637 0000E5D0 0000E5F0'                            .field repaste
A0638 0000E5F0      9561  repaste:                  move *pstk+, a1     ;rdx
A0639 0000E600      9560                            move *pstk+, a0     ;rdy
A0640 0000E610      0B81                            andi 0ffffh,a1
      0000E620 FFFF0000
A0641 0000E640      0B80                            andi 0ffffh,a0
      0000E650 FFFF0000
A0642 0000E670      2600                            sll 16,a0
A0643 0000E680      EE01                            movy a0,a1          ;(rdy,rdx)
A0644 0000E690      9563                            move *pstk+, a3     ;nx
A0645 0000E6A0      9562                            move *pstk+, a2     ;ny
A0646 0000E6B0      0B82                            andi 0ffffh,a2
      0000E6C0 FFFF0000
A0647 0000E6E0      0B83                            andi 0ffffh,a3
      0000E6F0 FFFF0000
A0648 0000E710      2602                            sll 16,a2
A0649 0000E720      EC62                            movx a3,a2          ;(nx,ny)
A0650 0000E730      8564                            move *pstk,a4
A0651 0000E740      9485                            move *a4+,a5        ;saddres
A0652 0000E750      9486                            move *a4+,a6        ;s(dx,dy)
A0653 0000E760      4C89                            move a4,a9
A0654 0000E770      9487                            move *a4+,a7        ;s(cpx,cpy)
A0655 0000E780      9488                            move *a4+,a8        ;sptch
A0656 0000E790      4EA4                            move a5,offset
A0657 0000E7A0      4F03                            move a8,dptch
A0658 0000E7B0      4EE0                            move a7,saddr
A0659 0000E7C0      4F01                            move a8,sptch
A0660 0000E7D0      4E27                            move a1,dydx    ;(rdy,rdx)
A0661 0000E7E0      8049                            move a2,*a9         ;atualiza cpx,cpy
A0662 0000E7F0      018A                            getst a10
A0663 0000E800      0550                            setf 16,0,0
A0664 0000E810      6AE9                            lmo a7,a9
A0665 0000E820      0589                            move a9,@convsp
      0000E830 C0000130
A0666 0000E850      0589                            move a9,@convdp
      0000E860 C0000140
A0667 0000E880      01AA                            putst a10
A0668 0000E890      0F60                            pixblt xy,xy
A0669 0000E8A0      0960                            rets
A0670                       *
A0671                       * adr adr dy dx TRANSFER -
A0672                       *      Source Dest  DY DX Transfer
```

```
A0673                          *       Transfer copia o stencil dydx apontado  por s(cpx,cpy) em
A0674                          *       d(cpx,cpy)
A0675                          *
A0676 0000E8B0 0000E560' h_transfer:                .int h_repaste
A0677 0000E8D0        08                            .byte 08,0
      0000E8D8        00
A0678 0000E8E0        54                            .string "TRANSFER"
      0000E8E8        52
      0000E8F0        41
      0000E8F8        4E
      0000E900        53
      0000E908        46
      0000E910        45
      0000E918        52
A0679 0000E920                                      .even
A0680 0000E920 0000E940'                            .field transfer
A0681 0000E940      9560  transfer:                 move *pstk+,a0   ;dx
A0682 0000E950      9561                            move *pstk+,a1   ;dy
A0683 0000E960      0B80                            andi 0ffffh,a0
      0000E970 FFFF0000
A0684 0000E990      0B81                            andi 0ffffh,a1
      0000E9A0 FFFF0000
A0685 0000E9C0      2601                            sll 16,a1
A0686 0000E9D0      EE20                            movy a1,a0       ;dydx
A0687 0000E9E0      9561                            move *pstk+,a1
A0688 0000E9F0      9422                            move *a1+,a2     ;daddr
A0689 0000EA00      9423                            move *a1+,a3     ;d(dy,dx)
A0690 0000EA10      9424                            move *a1+,a4     ;d(cpx,cpy)
A0691 0000EA20      9425                            move *a1+,a5     ;dpitch
A0692 0000EA30      4EA3                            move a5,dptch
A0693 0000EA40      4E44                            move a2,offset
A0694 0000EA50      018A                            getst a10
A0695 0000EA60      0550                            setf 16,0,0
A0696 0000EA70      6AA7                            lmo a5,a7        ;convdp
A0697 0000EA80      01AA                            putst a10
A0698 0000EA90      E886                            cvxyl a4,a6      ;destina 1 address
A0699 0000EAA0      9561                            move *pstk+,a1
A0700 0000EAB0      9422                            move *a1+,a2     ;addres
A0701 0000EAC0      9423                            move *a1+,a3     ;s(dy,dx)
A0702 0000EAD0      9424                            move *a1+,a4     ;s(cpx,cpy)
A0703 0000EAE0      9427                            move *a1+,a7     ;s(pitch)
A0704 0000EAF0      4EE3                            move a7,dptch
A0705 0000EB00      4E44                            move a2,offset
A0706 0000EB10      018A                            getst a10
A0707 0000EB20      0550                            setf 16,0,0
A0708 0000EB30      6AE8                            lmo a7,a8        ;convsp
A0709 0000EB40      01AA                            putst a10
A0710 0000EB50      E889                            cvxyl a4,a9      ;source 1 address
A0711 0000EB60      4E07                            move a0,dydx
A0712 0000EB70      4EA3                            move a5,dptch
A0713 0000EB80      4EC2                            move a6,daddr
A0714 0000EB90      4EE1                            move a7,sptch
A0715 0000EBA0      4F20                            move a9,saddr
A0716 0000EBB0      0F00                            pixblt 1,1
A0717 0000EBC0      0960                            rets
A0718                          *
A0719                          * adr GETCPT y x
A0720                          *       Source GETCPT
A0721                          *       Pega o cpx,cpy de source e coloca na pilha.
A0722                          *
A0723 0000EBD0 0000E8B0' h_getcpt:                  .int h_transfer
A0724 0000EBF0        06                            .byte 06,0h
      0000EBF8        00
A0725 0000EC00        47                            .string "GETCPT"
      0000EC08        45
      0000EC10        54
      0000EC18        43
      0000EC20        50
      0000EC28        54
A0726 0000EC30                                      .even
A0727 0000EC30 0000EC50'                            .field getcpt
A0728 0000EC50      9560  getcpt:                   move *pstk+,a0
A0729 0000EC60      0B00                            addi 32*2,a0         ;y,x
      0000EC70      0040
A0730 0000EC80      8401                            move *a0,a1          ;0
A0731 0000EC90      5642                            clr a2               ;-x
```

```
A0732 0000ECA0      EC22                          movx a1,a2
A0733 0000ECB0      2E01                          srl 16,a1
A0734 0000ECC0      0B81                          andi 0ffffh,a1 ;y
      0000ECD0 FFFF0000
A0735 0000ECF0      0B82                          andi 0ffffh,a2 ;x
      0000ED00 FFFF0000
A0736 0000ED20      A02B                          move a1,-*pstk
A0737 0000ED30      A04B                          move a2,-*pstk
A0738 0000ED40      0960                          rets
A0739                       *
A0740                       * y x adr  SETCPT -
A0741                       *      y x Source SETCPT -
A0742                       *      Atualiza S(CPX,CPY) para (X,Y)
A0743                       *
A0744 0000ED50 0000EBD0' h_setcpt:                .int h_getcpt
A0745 0000ED70        06                          .byte 06,0
      0000ED78        00
A0746 0000ED80        53                          .string "SETCPT"
      0000ED88        45
      0000ED90        54
      0000ED98        43
      0000EDA0        50
      0000EDA8        54
A0747 0000EDB0                                    .even
A0748 0000EDB0 0000EDD0'                          .field setcpt
A0749 0000EDD0      9560  setcpt:                 move *pstk+,a0
A0750 0000EDE0      9561                          move *pstk+,a1  ;x
A0751 0000EDF0      9562                          move *pstk+,a2  ;y
A0752 0000EE00      0B81                          andi 0ffffh,a1
      0000EE10 FFFF0000
A0753 0000EE30      0B82                          andi 0ffffh,a2
      0000EE40 FFFF0000
A0754 0000EE60      2602                          sll 16,a2
A0755 0000EE70      EC22                          movx a1,a2
A0756 0000EE80      0B00                          addI 64,a0
      0000EE90      0040
A0757 0000EEA0      8040                          move a2,*a0
A0758 0000EEB0      0960                          rets
A0759                       *
A0760                       * - CURSTENCIL adr
A0761                       *      Variavel que contem o endereco do STENCIL corrente onde
A0762                       *      as trajetorias serao aplicadas.
A0763                       *
A0764 0000EEC0 0000ED50' h_curstencil:            .int h_setcpt
A0765 0000EEE0        0A                          .byte 10,0
      0000EEE8        00
A0766 0000EEF0        43                          .string "CURSTENCIL"
      0000EEF8        55
      0000EF00        52
      0000EF08        53
      0000EF10        54
      0000EF18        45
      0000EF20        4E
      0000EF28        43
      0000EF30        49
      0000EF38        4C
A0767 0000EF40                                    .even
A0768 0000EF40 0000EF60'                          .field curstencil
A0769 0000EF60      0D5F  curstencil              calla docreate
      0000EF70 000010A0'
A0770 0000EF90 00000000  fcurstencil              .int 0h
A0771                       *
A0772                       * - CURBRUSH adr
A0773                       *      Variavel que contem o endereco do stencil  correspondente
A0774                       *      ao pincel adotado por "LINE"
A0775                       *
A0776 0000EFB0 0000EEC0' h_curbrush:              .int h_curstencil
A0777 0000EFD0        08                          .byte 08,0h
      0000EFD8        00
A0778 0000EFE0        43                          .string "CURBRUSH"
      0000EFE8        55
      0000EFF0        52
      0000EFF8        42
      0000F000        52
      0000F008        55
      0000F010        53
```

```
      0000F018        48
A0779 0000F020                                  .even
A0780 0000F020 0000F040'                        .field curbrush
A0781 0000F040      0D5F  curbrush:             calla docreate
      0000F050 000010A0'
A0782 0000F070 00000000  fcurbrush:             .int 0h
A0783                    *
A0784                    * y x LINEB -
A0785                    *      Traca linha do ponto corrente ate do Stencil XY, usando
A0786                    *      CURBRUSH
A0787                    *
A0788                    *      2 delta y - 2 delta x     a10
A0789                    *      2 delta y                 a9
A0790                    *      @ (2 delta y - delta x)   a8
A0791                    *      count ( delta x)          a7
A0792                    *      x y                       a6
A0793                    *      maior inc                 a5
A0794                    *      menor inc                 a4
A0795                    *      saddr (recomposicao)      a3
A0796                    *      daddr (recomposicao)      a2
A0797                    *
A0798 0000F090 0000EFB0' h_lineb:       .int h_curbrush
A0799 0000F0B0        05                .byte 05,0h
      0000F0B8        00
A0800 0000F0C0        4C                .string "LINEB"
      0000F0C8        49
      0000F0D0        4E
      0000F0D8        45
      0000F0E0        42
A0801 0000F0F0                          .even
A0802 0000F0F0 0000F110'                .field lineb
A0803 0000F110      05A0  lineb:        move @fcurbrush, a0     ;prepara para brush
      0000F120 0000F070'
A0804 0000F140      9401                move *a0+,a1            ;endereco inicial do brush
A0805 0000F150      4E20                move a1,saddr           ;armazena em b0
A0806 0000F160      9401                move *a0+,a1            ;pega dydx
A0807 0000F170      4E27                move a1,dydx            ;armazena em b7
A0808 0000F180      9402                move *a0+,a2            ;pega cpx,cpy (brush)
A0809 0000F190      4E48                move a2,b8
A0810 0000F1A0      9401                move *a0+,a1            ;pega pitch
A0811 0000F1B0      4E21                move a1, sptch          ;armazena em b1
A0812 0000F1C0      018A                getst a10               ;calcula @convsp
A0813 0000F1D0      0550                setf 16,0,0
A0814 0000F1E0      6A23                lmo a1,a3
A0815 0000F1F0      0583                move a3,@convsp
      0000F200 C0000130
A0816 0000F220      01AA                putst a10
A0817 0000F230      05A0                move @fcurstencil,a0    ;prepara para stencil
      0000F240 0000EF90'
A0818 0000F260      9401                move *a0+,a1            ;endereco inicial do stencial
A0819 0000F270      4E24                move a1,offset          ;origem do stencil
A0820 0000F280      9401                move *a0+,a1            ;pega dydx
A0821 0000F290      4E26                move a1,wend            ;coloca em wend
A0822 0000F2A0      9403                move *a0+,a3            ;*pega cpx,cpy (stencil)
A0823 0000F2B0      4E62                move a3,daddr
A0824 0000F2C0      9404                move *a0+,a4            ;pega pitch
A0825 0000F2D0      4E83                move a4,dptch           ;coloca em b3
A0826 0000F2E0      018A                getst a10               ;seta convdp e window
A0827 0000F2F0      0550                setf 16,0,0
A0828 0000F300      6A85                lmo a4,a5
A0829 0000F310      0585                move a5, @convdp
      0000F320 C0000140
A0830 0000F340      05A0                move @control,a0
      0000F350 C00000B0
A0831 0000F370      4E09                move a0,b9
A0832 0000F380      0BA0                ori 11000000b,a0        ;window modo 3
      0000F390 000000C0
A0833 0000F3B0      0580                move a0, @control
      0000F3C0 C00000B0
A0834 0000F3E0      01AA                putst a10
A0835 0000F3F0      5600                clr a0
A0836 0000F400      4E05                move  a0,wstart         ;coloca em b5
A0837 0000F410      9560  linebl:       move *pstk+,a0          ;pega xf da pilha
A0838 0000F420      9561                move *pstk+,a1          ;pega yf da pilha
A0839                    *              move a3,-*pstk          ;salva cpx,cpy stencil
A0840                    *              move a2,-*pstk          ;salva cpx,cpy brush
```

```
A0841 0000F430     0B80                 andi 0ffffh,a0          ;compoe xf,yf
      0000F440 FFFF0000
A0842 0000F460     0B81                 andi 0ffffh,a1
      0000F470 FFFF0000
A0843 0000F490     2601                 sll 16,a1
A0844 0000F4A0     EC01                 movx a0,a1              ;xf,yf em a1
A0845 0000F4B0     A02B                 move a1,-*pstk          ;salva xf,yf
A0846 0000F4C0     4C60  lineb2:        move a3,a0              ;a0 contem  xiyi
A0847 0000F4D0     E201                 subxy a0,a1             ;(xf,yf)-(xi-yi)
A0848 0000F4E0     5642                 clr a2
A0849 0000F4F0     EC22                 movx a1,a2
A0850 0000F500     2E01                 srl 16,a1               ;delta x  em a2
A0851 0000F510     0B81                 andi 0ffffh,a1           ;delta y  em a1
      0000F520 FFFF0000
A0852 0000F540     018A                 getst a10               ;extende sinal
A0853 0000F550     0550                 setf 16,0,0
A0854 0000F560     0501                 sext a1
A0855 0000F570     0502                 sext a2
A0856 0000F580     01AA                 putst a10               ;sign extended  delta y
A0857                   *                                       delta x
A0858 0000F590     56A5                 clr a5                  ;prepara maiorinc
A0859 0000F5A0     568A                 clr a4                  ;prepara menorinc
A0860 0000F5B0     0B42                 cmpi 0,a2
      0000F5C0     FFFF
A0861 0000F5D0     CA07                 jrz lineb4              ;se delta x + 0
A0862 0000F5E0     CE03                 jrn lineb3              ;se negativo
A0863 0000F5F0     09C5                 movi 1,a5               ;positivo maiorinc=1
      0000F600     0001
A0864 0000F610     C003                 jruc lineb4
A0865 0000F620     09E5  lineb3:        movi 0ffffh,a5           ;negativo maiorinc=-1
      0000F630 0000FFFF
A0866 0000F650     0B41  lineb4:        cmpi 0,a1
      0000F660     FFFF
A0867 0000F670     CA08                 jrz lineb6              ;se delta y = 0
A0868 0000F680     CE04                 jrn lineb5
A0869 0000F690     09E4                 movi 10000h,a4          ;menorinc = 1
      0000F6A0 00010000
A0870 0000F6C0     C003                 jruc lineb6
A0871 0000F6D0     09E4  lineb5:        movi 0ffff0000h,a4       ;menorinc=-1
      0000F6E0 FFFF0000
A0872 0000F700     0381  lineb6:        abs a1                  ;|delta y|
A0873 0000F710     0382                 abs a2                  ;|delta x|
A0874 0000F720     4841                 cmp a2,a1               ;|delta y| - |delta x|
A0875 0000F730     CE06                 jrn lineb7
A0876 0000F740     4C40                 move a2,a0              ;|delta x| < |delta y|
A0877 0000F750     4C22                 move a1,a2              ;troca delta x  com delta y
A0878 0000F760     4C01                 move a0,a1              ;
A0879 0000F770     4C80                 move a4,a0              ;troca maiorinc com
A0880                   *                                       menorinc
A0881 0000F780     4CA4                 move a5,a4
A0882 0000F790     4C05                 move a0,a5
A0883 0000F7A0     4C47  lineb7:        move a2,a7              ;delta maior em count
A0884 0000F7B0     0B47                 cmpi 0,a7
      0000F7C0     FFFF
A0885 0000F7D0     CB02                 jrnz lineb7a
A0886 0000F7E0     0B07                 addi 1,a7
      0000F7F0     0001
A0887 0000F800     4C29  lineb7a:       move a1,a9              ;calcula 2 delta y
A0888 0000F810     4129                 add a9,a9               ;
A0889 0000F820     4D28  lineb8:        move a9,a8        ;calcula 2 delta y - delta x = @
A0890 0000F830     4448                 sub a2,a8               ;
A0891 0000F840     4D0A  lineb9:        move a8,a10       ;calcula 2 delta y - 2 delta.x
A0892 0000F850     444A                 sub a2,a10              ;
A0893 0000F860     4E50  lineb10:       move daddr,a0           ;calculo do ponto x;y;
A0894 0000F870     4F11                 move b8,a1              ;a1 contem cpx,cpy brush
A0895 0000F880     E220                 subxy a1,a0               ;subtrai d(cpx,cpy)      de
A0896                   *                                       s(cpx,cpy)
A0897 0000F890     4C06                 move a0,a6              ;ponto inicial x,y
A0898 0000F8A0     4EC2                 move a6,daddr           ;
A0899 0000F8B0     4E13                 move saddr,a3           ;
A0900 0000F8C0     4E60  lineb11:       move a3,saddr           ;1 main loop,  reconstitui
A0901                   *                                       saddr
A0902 0000F8D0     4EC2                 move a6,daddr           ;reconstitui daddr
A0903 0000F8E0     0F20                 pixblt 1,xy             ;plot (x,y)
A0904 0000F8F0     0B48                 cmpi 0,a8               ;@ > 0 ?
      0000F900     FFFF
```

```
A0905 0000F910      CE03                jrn lineb12            ;a8-0
A0906 0000F920      E086                addxy a4,a6            ;xy + menorinc
A0907 0000F930      4148                add  a10,a8            ;
A0908 0000F940      C001                jruc lineb13
A0909 0000F950      4128  lineb12:      add a9,a8              ;@ = @ + 2 delta y
A0910 0000F960      E0A6  lineb13:      addxy a5,a6            ;xy + maiorinc
A0911 0000F970      3D87                dsj a7,lineb11         ;count = count - 1
A0912 0000F980      4E60                move a3,saddr
A0913 0000F990      4EC2                move a6,daddr
A0914 0000F9A0      0F20                pixblt 1,xy            ;ultimo ponto
A0915 0000F9B0      9560                move *patk+,a0         ;xfyf
A0916 0000F9C0      05A1                move @fcurstencil,a1   ;
      0000F9D0 0000EF90'
A0917 0000F9F0      9422                move *a1+ ,a2          ;endereco de stencil
A0918 0000FA00      9422                move *a1+,a2           ;dydx
A0919 0000FA10      8001                move a0,*a1            ;armazena em curstencil em
A0920                         *                                 cpx,cpy o valor xfyf
A0921 0000FA20      018A                getst a10              ;restabelece @control
A0922 0000FA30      0550                setf 16,0,0
A0923 0000FA40      0599                move b9,@control
      0000FA50 C0000080
A0924 0000FA70      01AA                putst r10
A0925 0000FA80      0960                rets
A0926                         *
A0927                         * - BOUNDARY adr
A0928                         *      Contem  valor do Pixel que sera utilizado em FILL (TRUE = 1,
A0929                         *      FALSE = 0) 1 bit.
A0930                         *
A0931 0000FA90 0000F090' h_boundary:     .int h_lineb
A0932 0000FAB0        08                 .byte 08,0
      0000FAB8        00
A0933 0000FAC0        42                 .string "BOUNDARY"
      0000FAC8        4F
      0000FAD0        55
      0000FAD8        4E
      0000FAE0        44
      0000FAE8        41
      0000FAF0        52
      0000FAF8        59
A0934 0000FB00                           .even
A0935 0000FB00 0000FB20'                 .field boundary
A0936 0000FB20      0D5F  boundary:      calla docreate
      0000FB30 000010A0'
A0937 0000FB50 FFFFFFFF  fboundary:      .int true
A0938                         *
A0939                         * true yx yx ... SFILL -
A0940                         *      Rotina SEEDFILL  chamada por FILL Na pilha  estao
A0941                         *      os Seed Points em formato xy e a marca do fim de Seed (True).
A0942                         *      Curstencil, Transparency e Loadfunction Validos.
A0943                         *
A0944 0000FB70 0000FA90' h_sfill:        .int h_boundary
A0945 0000FB90        05                 .byte 05,0h
      0000FB98        00
A0946 0000FBA0        53                 .string "SFILL"
      0000FBA8        46
      0000FBB0        49
      0000FBB8        4C
      0000FBC0        4C
A0947 0000FBD0                           .even
A0948 0000FBD0 0000FBF0'                 .field sfill
A0949 0000FBF0      05A0  sfill:         move @fcurstencil, A0        ;prepara stencil
      0000FC00 0000EF90'
A0950 0000FC20      9401                 move *a0+,a1                  ;endereco,marca
A0951 0000FC30      4E24                 move a1, offset               ;origem stencil
A0952 0000FC40      9401                 move *a0+,a1                  ;dydx
A0953 0000FC50      4E26                 move a1,wend                  ;window end
A0954 0000FC60      9401                 move *a0+,a1                  ;cpx,cpy
A0955 0000FC70      9401                 move *a0+,a1                  ;pitch
A0956 0000FC80      4E23                 move a1,dptch                 ;dest.pitch
A0957 0000FC90      018A                 getst a10                     ;prepara conusp
A0958 0000FCA0      0550                 setf 16,0,0
A0959 0000FCB0      6A22                 lmo a1,a2
A0960 0000FCC0      0582                 move a2,@convsp
      0000FCD0 C0000130
A0961 0000FCF0      0582                 move a2,@convdp
      0000FD00 C0000140
```

```
A0962 0000FD20     05A3                 move @control,a3              ;set a window
      0000FD30 C00000B0
A0963 0000FD50     4E68                 move a3,b8
A0964 0000FD60     0BA3                 ori 11000000b,a3             ;made 3
      0000FD70 000000C0
A0965 0000FD90     0583                 move a3,@control
      0000FDA0 C00000B0
A0966 0000FDC0     01AA                 putst a10
A0967 0000FDD0     5663                 clr a3
A0968 0000FDE0     4E65                 move a3,wstart               ;window start
A0969 0000FDF0     05AA                 move @fboundary,a10
      0000FE00 0000FB50'
A0970 0000FE20     4F49                 move a10,b9                  ;color
A0971 0000FE30     0B8A                 andi 1,a10
      0000FE40 FFFFFFFE
A0972 0000FE60     9565  sfill1:        move *pstk+,a5               ;pop pixel (x,y)
A0973 0000FE70     0B45  sfill2:        cmpi true, a5                ;while pixe = wark
      0000FE80     0000
A0974 0000FE90     CA63                 jrz sfill14                  ;se = termina
A0975 0000FEA0     F145                 pixt a10,*a5.xy      ;pixel(x,y)-fill value
A0976 0000FEB0     ECA7                 movx a5,a7                   ;savex=x
A0977 0000FEC0     09C4                 movi  1,a4
      0000FED0     0001
A0978 0000FEE0     E085                 addxy a4,a5
A0979 0000FEF0     5621  sfill3:        clr a1               ;while pixel(x,y)=bound
A0980 0000FF00     F2A1                 pixt *a5.xy,a1
A0981 0000FF10     0B81                 andi 01,a1
      0000FF20 FFFFFFFE
A0982 0000FF40     4941                 cmp a10,a1
A0983 0000FF50     CA02                 jrz sfill4
A0984 0000FF60     F685                 drav  a4,a5          ;pixel (x,y)-fill value
A0985 0000FF70     CDF7                 jrnv sfill3         ;x=x+1, window value
A0986 0000FF80     09E4  sfill4:        movi 0ffffh,a4               ;
      0000FF90 0000FFFF
A0987 0000FFB0     E085                 addxy a4,a5                  ;x=x-1
A0988 0000FFC0     ECA9                 movx a5,a9                   ;xright - x
A0989 0000FFD0     ECE5                 movx a7,a5                   ;x - savex
A0990 0000FFE0     E085                 addxy a4,a5                  ;x=x-1
A0991 0000FFF0     5621  sfill5:        clr a1               ;while pixel (x,y) = bound
A0992 00010000     F2A1                 pixt *a5.xy,a1
A0993 00010010     0B81                 andi 01,a1
      00010020 FFFFFFFE
A0994 00010040     4941                 cmp a10,a1
A0995 00010050     CA02                 jrz sfill6
A0996 00010060     F685                 drav a4,a5
A0997 00010070     CDF7                 jrnv sfill5                  ;window violation
A0998 00010080     09C4  sfill6:        movi 1,a4                    ;xinc=+1
      00010090     0001
A0999 000100A0     E085                 addxy a4,a5                  ;x+x+1
A1000 000100B0     ECA8                 movx a5,a8                   ;xleft - x
A1001 000100C0     EEA6                 movy a5,a6                   ;savey - y.xleft
A1002 000100D0     09E3                 movi 10000h,a3               ;yinc
      000100E0 00010000
A1003 00010100     E065                 addxy a3,a5                  ;y=y+1, xleft
A1004 00010110     E4A9  sfill7:        cmpxy a5,a9                  ;x<xr16hit
A1005 00010120     CE19                 jrn sfill10
A1006 00010130     CC18                 jrv sfill10
A1007 00010140     5621                 clr a1
A1008 00010150     F2A1                 pixt *a5.xy,a1       ;pixel (x,y) -  bound
A1009 00010160     E085                 addxy a4,a5
A1010 00010170     0B81                 andi 01,a1
      00010180 FFFFFFFE
A1011 000101A0     4941                 cmp a10,a1
A1012 000101B0     CAF5                 jrz sfill7
A1013 000101C0     5621  sfill8:        clr a1               ;while pixel(x,y) = bound
A1014 000101D0     F2A1                 pixt *a5.xy,a1
A1015 000101E0     0B81                 andi 01,a1
      000101F0 FFFFFFFE
A1016 00010210     4941                 cmp a10,a1
A1017 00010220     CA02                 jrz sfill9
A1018 00010230     E085                 addxy a4,a5                  ;x+x=1
A1019 00010240     C0F7                 jruc sfill8
A1020 00010250     09E0  sfill9:        movi 0ffffh,a0
      00010260 0000FFFF
A1021 00010280     E005                 addxy a0,a5                  ;pixel (x-1,y)
A1022 00010290     A0AB                 move a5,-*pstk       ;push pixel (x+1,y)
```

```
A1023 000102A0      E085                addxy a4,a5
A1024 000102B0      C0E5                jruc sfill7
A1025 000102C0      ED05  sfill10:      movx a8,a5                    ;x=xleft
A1026 000102D0      EEC5                movy a6,a5
A1027 000102E0      09E3                movi 0ffff0000h, a3           ;
      000102F0 FFFF0000
A1028 00010310      E065                addxy a3,a5                   ;y=y-1
A1029 00010320      E4A9  sfill11:      cmpxy a5,a9                   ;x<xright
A1030 00010330      CEB2                jrn sfill1
A1031 00010340      CCB1                jrv sfill1
A1032 00010350      5621                clr a1
A1033 00010360      F2A1                pixt *a5.xy,a1        ;pixel (x,y) = bound ?
A1034 00010370      E085                addxy a4,a5
A1035 00010380      0B81                andi 01,a1
      00010390 FFFFFFFE
A1036 000103B0      4941                cmp a10,a1
A1037 000103C0      CAF5                jrz sfill11
A1038 000103D0      5621  sfill12:      clr a1                ;while pixel (x,y) = bound
A1039 000103E0      F2A1                pixt *a5.xy,a1
A1040 000103F0      0B81                andi 01,a1
      00010400 FFFFFFFE
A1041 00010420      4941                cmp a10,a1
A1042 00010430      CA02                jrz sfill13
A1043 00010440      E085                addxy a4,a5
A1044 00010450      C0F7                jruc sfill12
A1045 00010460      09E0  sfill13:      movi 0ffffh, a0
      00010470 0000FFFF
A1046 00010490      E005                addxy a0,a5
A1047 000104A0      A0AB                move a5,-*pstk
A1048 000104B0      E085                addxy a4,a5
A1049 000104C0      C0E5                jruc sfill11
A1050 000104D0            sfill14:      ;reestabelece control
A1051 000104D0      018A                getst a10
A1052 000104E0      0550                setf 16,0,0
A1053 000104F0      0598                move b8,@control
      00010500 C00000B0
A1054 00010520      01AA                putst a10
A1055 00010530      0960                rets
A1056                     *
A1057                     * true y x y x ... FILL -
A1058                     *      Pega as "sementes" ate flag true  converte para modo xy e chama
A1059                     *      a rotina sfill.
A1060                     *
A1061 00010540 0000FB70'  h_fill:     .int n_sfill
A1062 00010560        04              .byte 04,0
      00010568        00
A1063 00010570        46              .string "FILL"
      00010578        49
      00010580        4C
      00010588        4C
A1064 00010590                        .even
A1065 00010590 000105B0'              .field fill
A1066 000105B0      9560  fill:       move *pstk+,a0
A1067 000105C0      0B40              cmpi true,a0
      000105D0      0000
A1068 000105E0      CA15              jrz fill1
A1069 000105F0      9561              move *pstk+,a1
A1070 00010600      0B41              cmpi true, a1
      00010610      0000
A1071 00010620      CA11              jrz fill1
A1072 00010630      0B80              andi 0ffffh,a0                ;x
      00010640 FFFF0000
A1073 00010660      0B81              andi 0ffffh,a1                ;y
      00010670 FFFF0000
A1074 00010690      2601              sll 16,a1
A1075 000106A0      EC01              movx a0,a1
A1076 000106B0      09E3              movi true,a3
      000106C0 FFFFFFFF
A1077 000106E0      A06B              move a3,-*pstk
A1078 000106F0      A02B              move a1,-*pstk
A1079 00010700      0D5F              calla sfill
      00010710 0000FBF0'
A1080 00010730      C0E7              jruc fill
A1081 00010740      0960  fill1:      rets
A1082                     *
A1083                     * y x LINEF -
```

```
A1084                          *       Produz o outline  a ser utilizado por fill.
A1085                          *       XY Linef.
A1086                          *       Desenha a linha ate xy, usando boundary
A1087                          *
A1088 00010750 00010540' h_linef:        .int h_fill
A1089 00010770       05                  .byte 05,0h
      00010778       00
A1090 00010780       4C                  .string "LINEF"
      00010788       49
      00010790       4E
      00010798       45
      000107A0       46
A1091 000107B0                           .even
A1092 000107B0 000107D0'                 .field linef
A1093 000107D0       05A0  linef:        move @fcurstencil,a0          ;prepara stencil
      000107E0 0000EF90'
A1094 00010800       9401                move *a0+,a1
A1095 00010810       4E24                move a1,offset                ;endereco inicial
A1096 00010820       9401                move *a0+,a1                  ;dydx
A1097 00010830       4E26                move a1, wend                 ;
A1098 00010840       4E08                move a0,b8
A1099 00010850       9402                move *a0+,a2                  ;cpy,cpx
A1100 00010860       4E42                move a2, daddr                ;starting point
A1101 00010870       56B5                clr wstart                    ;starting corner
A1102 00010880       9403                move *a0+,a3                  ;pitch
A1103 00010890       09FD                movi true, pattrn
      000108A0 FFFFFFFF
A1104 000108C0       4E63                move a3,dptch
A1105 000108D0       018A                getst a10
A1106 000108E0       0550                setf 16,0,0
A1107 000108F0       6A64                lmo a3,a4
A1108 00010900       0584                move a4,@convdp
      00010910 C0000140
A1109 00010930       05A4                move @control, a4
      00010940 C00000B0
A1110 00010960       4E81                move a4,b1
A1111 00010970       0BA0                ori 11000000b,a0
      00010980 000000C0
A1112 000109A0       0584                move a4, @control
      000109B0 C00000B0
A1113 000109D0       01AA                putst a10
A1114 000109E0       05B9                move @fboundary, color1
      000109F0 0000FB50'
A1115 00010A10       9560  linef1:       move *pstk+,a0                ;xf
A1116 00010A20       9561                move *pstk+,a1                ;yf
A1117 00010A30       0B80                andi 0ffffh,a0
      00010A40 FFFF0000
A1118 00010A60       0B81                andi 0ffffh,a1
      00010A70 FFFF0000
A1119 00010A90       2601                sll 16,a1
A1120 00010AA0       EC01                movx a0,a1                    ;yfxf ou a1
A1121 00010AB0       A02B  linef2:       move a1,-*pstk
A1122 00010AC0       E241                subxy a2,a1
A1123 00010AD0       5600                clr a0
A1124 00010AE0       EC20                movx a1,a0                    ;x
A1125 00010AF0       2E01                srl 16,a1
A1126 00010B00       0B81                andi 0ffffh,a1
      00010B10 FFFF0000
A1127 00010B30       018A                getst a10
A1128 00010B40       0550                setf 16,0,0
A1129 00010B50       0501                sext a1                       ;y
A1130 00010B60       0500                sext a0                       ;x
A1131 00010B70       01AA                putst a10
A1132 00010B80       577B                clr inc1                      ;y
A1133 00010B90       579C                clr inc2                      ;x
A1134 00010BA0       0B40                cmpi 0,a0
      00010BB0       FFFF
A1135 00010BC0       CA07                jrz linef4
A1136 00010BD0       CE03                jrn linef3
A1137 00010BE0       09DC                movi 1, inc2
      00010BF0       0001
A1138 00010C00       C003                jruc linef4
A1139 00010C10       09FC  linef3:       movi 0ffffh,inc2
      00010C20 0000FFFF
A1140 00010C40       0B41  linef4:       cmpi 0,a1
      00010C50       FFFF
```

```
A1141 00010C60      CA08                jrz linef6
A1142 00010C70      CE04                jrn linef5
A1143 00010C80      09FB                movi 10000h,incl
      00010C90 00010000
A1144 00010CB0      C003                jruc linef6
A1145 00010CC0      09FB  linef5:       movi 0ffff0000h, incl
      00010CD0 FFFF0000
A1146 00010CF0      0380  linef6:       abs a0                          ;|delta x|
A1147 00010D00      0381                abs a1                          ;|delta y|
A1148 00010D10      4801                cmp a0,a1             ;|delta y| - |delta x|
A1149 00010D20      CE06                jrn linef7
A1150 00010D30      4C22                move a1,a2            ;|delta x|<|delta y|
A1151 00010D40      4C01                move a0, a1           ;troca delta x com delta y
A1152 00010D50      4C40                move a2, a0
A1153 00010D60      4F72                move incl,a2          ;troca incl com inc2
A1154 00010D70      4D9B                move inc2,incl
A1155 00010D80      4E4C                move a2,inc2
A1156 00010D90      559B  linef7:             or inc2,incl
A1157 00010DA0      4E0A                move a0,count         ;delta maior em count
A1158 00010DB0      103A                addk 1, count
A1159 00010DC0      4C02                move a0,a2
A1160 00010DD0      4C29                move a1,a9
A1161 00010DE0      2609                sll 16,a9
A1162 00010DF0      EF22                movy a9,a2
A1163 00010E00      4E47                move a2,dydx                    ;(b,a)
A1164 00010E10      4C22                move a1,a2
A1165 00010E20      4042                add a2,a2
A1166 00010E30      4402                sub a0,a2             ;2 delta y - delta x
A1167 00010E40      4E40                move a2, saddr
A1168 00010E50      DF1A                line 0
A1169 00010E60      9560                move *pstk+,a0        ;atualiza cpx,cpy
A1170 00010E70      4F11                move b8, a1
A1171 00010E80      8001                move a0, *a1
A1172 00010E90      018A                getst a10
A1173 00010EA0      0550                setf 16,0,0
A1174 00010EB0      0591                move b1, @control
      00010EC0 C00000B0
A1175 00010EE0      01AA                putst a10
A1176 00010EF0      0960                rets
A1177                         *
A1178                         * - ROTAVAR adr
A1179                         *
A1180                         *       Varialvel de controle.
A1181                         *       Contem o angulo em  graus que  deve ser rotacionada
A1182                         *       as trajetorias.
A1183                         *       Recomenda-se nao ultrapassar 180 graus.
A1184                         *
A1185 00010F00 00010750' h_rotavar:     .int h_linef
A1186 00010F20       07                 .byte 07,0
      00010F28       00
A1187 00010F30       52                 .string "ROTAVAR"
      00010F38       4F
      00010F40       54
      00010F48       41
      00010F50       56
      00010F58       41
      00010F60       52
A1188 00010F70                          .even
A1189 00010F70 00010F90'                .field rotavar
A1190 00010F90      0D5F  rotavar:      calla docreate
      00010FA0 000010A0'
A1191 00010FC0 00000000  frotation:     .int 0
A1192                         *
A1193 00010FE0 00010F00' h_rotation:    .int h_rotavar
A1194 00011000       08                 .byte 08,0
      00011008       00
A1195 00011010       52                 .string "ROTATION"
      00011018       4F
      00011020       54
      00011028       41
      00011030       54
      00011038       49
      00011040       4F
      00011048       4E
A1196 00011050                          .even
A1197 00011050 00011070'                .field rotation
```

```
A1198 00011070      0D5F  rotation:      calla rotavar
      00011080 00010F90'
A1199 000110A0      0D5F                 calla store
      000110B0 00000CB0'
A1200 000110D0      0960                 rets
A1201                     *
A1202                     * y x RROTATION y x
A1203                     *
A1204                     *      Rotina de rotacao.
A1205                     *      Pega os pontos xy e rotaciona com relacao a 0,0 ,o angulo
A1206                     *      em rotation.
A1207                     *
A1208 000110E0 00010FE0' h_rrotation:   .int h_rotation
A1209 00011100        09                .byte 09,0
      00011108        00
A1210 00011110        52                .string "RROTATION"
      00011118        52
      00011120        4F
      00011128        54
      00011130        41
      00011138        54
      00011140        49
      00011148        4F
      00011150        4E
A1211 00011160                          .even
A1212 00011160 00011180'                .field rrotation
A1213 00011180      05A8  rrotation:     move @frotation,a8
      00011190 00010FC0'
A1214 000111B0      0B48                 cmpi 0,a8
      000111C0      FFFF
A1215 000111D0      CA31                 jrz rrotation6
A1216 000111E0      C105                 jrp rrotation1      ;se negativo inverte x-y
A1217 000111F0      5729                 clr a9
A1218 00011200      0388                 abs a8
A1219 00011210      9562                 move *pstk+, a2                ;x
A1220 00011220      9560                 move *pstk+,a0                 ;y
A1221 00011230      C005                 jruc rrotation2
A1222 00011240      09E9  rrotation1:    movi true,a9
      00011250 FFFFFFFF
A1223 00011270      9560                 move *pstk+,a0                 ;x
A1224 00011280      9562                 move *pstk+,a2                 ;y
A1225 00011290      018A  rrotation2:    getst a10
A1226 000112A0      0740                 setf 32,0,1
A1227 000112B0      09C7                 movi 143,a7
      000112C0      008F
A1228 000112D0      09C6                 movi 8192,a6
      000112E0      2000
A1229 000112F0      5CC2                 mpys a6,a2
A1230 00011300      4C62                 move a3,a2
A1231 00011310      5CC0                 mpys a6,a0
A1232 00011320      4C20                 move a1,a0
A1233 00011330      4C44  rrotation3:    move a2,a4                     ;y
A1234 00011340      5CE4                 mpys a7,a4                     ;y.143
A1235 00011350      58C4                 divs a6,a4                     ;y.143/8192
A1236 00011360      4C01                 move a0,a1
A1237 00011370      4081                 add a4,a1             ;x'=x+y.143/8192
A1238 00011380      4C04                 move a0,a4
A1239 00011390      5CE4                 mpys a7,a4
A1240 000113A0      58C4                 divs a6,a4
A1241 000113B0      4C43                 move a2,a3
A1242 000113C0      4483                 sub a4,a3           ;y'=y-x.143/8192
A1243 000113D0      4C20                 move a1,a0
A1244 000113E0      4C62                 move a3,a2
A1245 000113F0      3DA8                 dsj a8,rrotation3
A1246 00011400      4C43                 move a2,a3
A1247 00011410      4C01                 move a0,a1
A1248 00011420      5642                 clr a2
A1249 00011430      5600                 clr a0
A1250 00011440      58C2                 divs a6,a2
A1251 00011450      58C0                 divs a6,a0
A1252 00011460      01AA                 putst a10
A1253 00011470      0B49                 cmpi true,a9
      00011480      0000
A1254 00011490      CA03                 jrz rrotation5
A1255 000114A0      A00B  rrotation4:    move a0,-*pstk
A1256 000114B0      A04B                 move a2,-*pstk
```

```
A1257 000114C0      C002                    jruc rrotation6
A1258 000114D0      A04B  rrotation5:       move a2,-*pstk
A1259 000114E0      A00B                    move a0,-*pstk
A1260 000114F0      0960  rrotation6:       rets
A1261                     *
A1262                     *  - DOUBREVAR name
A1263                     *      Cria uma variavel dupla de valores.
A1264                     *      Visa a armazenar pares xy, escalas, etc. Inicializado em 0,1.
A1265                     *      Quando nome e invocado, retorna enderecos na pilha.
A1266                     *      DOUBLEVAR - nome.
A1267                     *
A1268 00011500 000110E0' h_doublevar:       .int h_rrotation
A1269 00011520        09                    .byte 09,0
      00011528        00
A1270 00011530        44                    .string "DOUBLEVAR"
      00011538        4F
      00011540        55
      00011548        42
      00011550        4C
      00011558        45
      00011560        56
      00011568        41
      00011570        52
A1271 00011580                              .even
A1272 00011580 000115A0'                    .field doublevar
A1273 000115A0      0D5F  doublevar:        calla builds        ;(deixa here na pilha)
      000115B0 00003EE0'
A1274 000115D0      05A5                    move @fdp,a5        ;contem pointer dic.
      000115E0 00001810'
A1275 00011600      09C0                    movi 0,a0
      00011610      0000
A1276 00011620      9005                    move a0,*a5+
A1277 00011630      1020                    addk 1,a0
A1278 00011640      9005                    move a0,*a5+
A1279 00011650      0585                    move a5,@fdp
      00011660 00001810'
A1280 00011680      0D5F                    calla does
      00011690 000040D0'
A1281 000116B0      95E0  rdoublevar:       move *sp+,a0
A1282 000116C0      A00B                    move a0,-*pstk
A1283 000116D0      0960                    rets
A1284                     *
A1285                     * adr DOUBLEFETCH y x
A1286                     *      Fetch para Doublevar.
A1287                     *
A1288 000116E0 00011500' h_doublefetch: .int h_doublevar
A1289 00011700        02                    .byte 2,0
      00011708        00
A1290 00011710        44                    .string "D@"
      00011718        40
A1291 00011720                              .even
A1292 00011720 00011740'                    .field doublefetch
A1293 00011740      9561  doublefetch:      move *pstk+,a1
A1294 00011750      9422                    move *a1+,a2                ;y
A1295 00011760      8423                    move *a1,a3                 ;x
A1296 00011770      A04B                    move a2,-*pstk              ;y
A1297 00011780      A06B                    move a3,-*pstk              ;x
A1298 00011790      0960                    rets
A1299                     *
A1300                     * y x DDUP y x y x
A1301                     *      funcao auxiliar , duplica uma variavel dupla
A1302                     *
A1303 000117A0 000116E0' H_DDUP             .int H_doublefetch
A1304 000117C0        04                    .byte 04,0
      000117C8        00
A1305 000117D0        44                    .string "DDUP"
      000117D8        44
      000117E0        55
      000117E8        50
A1306 000117F0                              .even
A1307 000117F0 00011810'                    .field ddup
A1308 00011810      9560  ddup:             move *pstk+,a0
A1309 00011820      9561                    move *pstk+,a1
A1310 00011830      A02B                    move a1,-*pstk
A1311 00011840      A00B                    move a0,-*pstk
A1312 00011850      A02B                    move a1,-*pstk
```

```
A1313 00011860      A00B                    move a0,-*pstk
A1314 00011870      0960                    rets
A1315                        *
A1316                        * y x adr DOUBLESTORE -
A1317                        *      Store para Doublevar.
A1318                        *      Y  X  end D!
A1319                        *
A1320 00011880 000117A0' h_doublestore: .int h_ddup
A1321 000118A0        02                 .byte 02,0
      000118A8        00
A1322 000118B0        44                 .string "D!"
      000118B8        21
A1323 000118C0                           .even
A1324 000118C0 000118E0'                 .field doublestore
A1325 000118E0      9561  doublestore:   move *pstk+,a1
A1326 000118F0      9562                 move *pstk+,a2                ;x
A1327 00011900      9563                 move *pstk+,a3                ;y
A1328 00011910      9061                 move a3,*a1+                  ;y
A1329 00011920      9041                 move a2,*a1+                  ;x
A1330 00011930      0960                 rets
A1331                        *
A1332                        * - VCPT adr
A1333                        *      Virtual Current point
A1334                        *      variavel de controle dupla que armazena o ponto corrente
A1335                        *      no sistema virtual de coordenadas
A1336                        *
A1337 00011940 00011880' H_VCPT:                .int H_doublestore
A1338 00011960        04                 .byte 04,0
      00011968        00
A1339 00011970        56                 .string "VCPT"
      00011978        43
      00011980        50
      00011988        54
A1340 00011990                           .even
A1341 00011990 000119B0'                 .field VCPT
A1342 000119B0      0D5F  vcpt:          calla rdoublevar
      000119C0 000116B0'
A1343 000119E0 00000000                  .int 0
A1344 00011A00 00000000                  .int 0
A1345                        *
A1346                        * - SCALEX adr
A1347                        *      Variavel de controle duplo   Contem a escala a/b que sera
A1348                        *      usada para escalar o eixo x:
A1349                        *      X' = x.a/b
A1350                        *
A1351 00011A20 00011940' h_scalex:       .int h_vcpt
A1352 00011A40        06                 .byte 06,0
      00011A48        00
A1353 00011A50        53                 .string "SCALEX"
      00011A58        43
      00011A60        41
      00011A68        4C
      00011A70        45
      00011A78        58
A1354 00011A80                           .even
A1355 00011A80 00011AA0'                 .field scalex
A1356 00011AA0      0D5F  scalex:        calla rdoublevar
      00011AB0 000116B0'
A1357 00011AD0 00000001  fscalexa:       .int 1                        ;y
A1358 00011AF0 00000001  fscalexb:       .int 1                        ;x
A1359                        *
A1360                        * - SHEARX adr
A1361                        *      Variavel de controle duplo Contem a escala a/b que sera
A1362                        *      usada na transformacao:
A1363                        *      X' = x + ya/b
A1364                        *
A1365 00011B10 00011A20' h_shearx:       .int h_scalex
A1366 00011B30        06                 .byte 06,0
      00011B38        00
A1367 00011B40        53                 .string "SHEARX"
      00011B48        48
      00011B50        45
      00011B58        41
      00011B60        52
      00011B68        58
A1368 00011B70                           .even
```

```
A1369 00011B70 00011B90'                 .field shearx
A1370 00011B90      0D5F  shearx:        calla rdoublevar
      00011BA0 000116B0'
A1371 00011BC0 00000000  fshearxa:       .int 0
A1372 00011BE0 00000001  fshearxb:       .int 1
A1373                    *
A1374                    * - SCALEY adr
A1375                    *       Variavel de controle duplo Contem a escala a/b   que sera
A1376                    *       usada para escalar o eixo y:
A1377                    *       Y' = y.a/b
A1378                    *
A1379 00011C00 00011B10' h_scaley:       .int h_shearx
A1380 00011C20        06                 .byte 06,0
      00011C28        00
A1381 00011C30        53                 .string "SCALEY"
      00011C38        43
      00011C40        41
      00011C48        4C
      00011C50        45
      00011C58        59
A1382 00011C60                           .even
A1383 00011C60 00011C80'                 .field scaley
A1384 00011C80      0D5F  scaley:        calla rdoublevar
      00011C90 000116B0'
A1385 00011CB0 00000001  fscaleya:       .int 1                          ;y
A1386 00011CD0 00000001  fscaleyb:       .int 1                          ;x
A1387                    *
A1388                    * - SHEARY adr
A1389                    *       Variavel de controle duplo Contem a escala a/b que sera
A1390                    *       usada na transformacao:
A1391                    *       Y' = y + x.a/b
A1392                    *
A1393 00011CF0 00011C00' h_sheary:       .int h_scaley
A1394 00011D10        06                 .byte 06,0
      00011D18        00
A1395 00011D20        53                 .string "SHEARY"
      00011D28        48
      00011D30        45
      00011D38        41
      00011D40        52
      00011D48        59
A1396 00011D50                           .even
A1397 00011D50 00011D70'                 .field sheary
A1398 00011D70      0D5F  sheary:        calla rdoublevar
      00011D80 000116B0'
A1399 00011DA0 00000000  fshearya:       .int 0
A1400 00011DC0 00000001  fshearyb:       .int 1
A1401                    *
A1402                    * y x SCALE y x
A1403                    *       Transforma os pontos Y X de acordo com Scalex Scaley.
A1404                    *
A1405 00011DE0 00011CF0' h_scale:        .int h_sheary
A1406 00011E00        05                 .byte 05,0
      00011E08        00
A1407 00011E10        53                 .string "SCALE"
      00011E18        43
      00011E20        41
      00011E28        4C
      00011E30        45
A1408 00011E40                           .even
A1409 00011E40 00011E60'                 .field scale
A1410 00011E60      9560  scale:         move *pstk+,a0                  ;x
A1411 00011E70      9561                 move *pstk+,a1                  ;y
A1412 00011E80      05A4                 move @fscalexa,a4
      00011E90 00011AD0'
A1413 00011EB0      05A6                 move @fscalexb,a6
      00011EC0 00011AF0'
A1414 00011EE0      018A                 getst a10
A1415 00011EF0      0740                 setf 32,0,1
A1416 00011F00      5C04                 mpys a0,a4
A1417 00011F10      58C4                 divs a6,a4                      ;x'  em a4
A1418 00011F20      05A6                 move @fscaleya,a6
      00011F30 00011CB0'
A1419 00011F50      05A8                 move @fscaleyb,a8
      00011F60 00011CD0'
A1420 00011F80      5C26                 mpys a1,a6
```

```
A1421 00011F90      5906                divs a8,a6                  ;y'  em a6
A1422 00011FA0      01AA                putst a10
A1423 00011FB0      A0CB                move a6,-*pstk
A1424 00011FC0      A08B                move a4,-*pstk
A1425 00011FD0      0960                rets
A1426                       *
A1427                       * y x SHEAR y x
A1428                       *      Transforma os pontos Y X , de acordo com Shearx Sheary
A1429                       *
A1430 00011FE0 00011DE0' h_shear:       .int h_scale
A1431 00012000        05                .byte 05,0
      00012008        00
A1432 00012010        53                .string "SHEAR"
      00012018        48
      00012020        45
      00012028        41
      00012030        52
A1433 00012040                          .even
A1434 00012040 00012060'                .field shear
A1435 00012060      9560  shear:        move *pstk+,a0              ;x
A1436 00012070      9561                move *pstk+,a1              ;y
A1437 00012080      018A                getst a10
A1438 00012090      0740                setf 32,0,1
A1439 000120A0      05A4                move @fshearxa,a4
      000120B0 00011BC0'
A1440 000120D0      05A6                move @fshearxb,a6
      000120E0 00011BE0'
A1441 00012100      5C24                mpys a1,a4
A1442 00012110      58C4                divs a6,a4
A1443 00012120      4004                add a0,a4
A1444 00012130      4C82                move a4,a2                  ;x'
A1445 00012140      05A4                move @fshearya,a4
      00012150 00011DA0'
A1446 00012170      05A6                move @fshearyb,a6
      00012180 00011DC0'
A1447 000121A0      5C04                mpys a0,a4
A1448 000121B0      58C4                divs a6,a4
A1449 000121C0      4024                add a1,a4
A1450 000121D0      4C83                move a4,a3                  ;y'
A1451 000121E0      01AA                putst a10
A1452 000121F0      A06B                move a3,-*pstk              ;y'
A1453 00012200      A04B                move a2,-*pstk              ;x'
A1454 00012210      0960                rets
A1455                       *
A1456                       * - GRAPHMODE adr
A1457                       *      Variavel  de controle. Se true = "DRAW MODE"
A1458                       *                             Se false = "FILL MODE"
A1459                       *      Especifica a acao dos comandos:
A1460                       *      Lineto
A1461                       *      Rlineto
A1462                       *      Moveto
A1463                       *      Rmoveto
A1464                       *
A1465 00012220 00011FE0' h_graphmode:   .int h_shear
A1466 00012240        09                .byte 09,0
      00012248        00
A1467 00012250        47                .string "GRAPHMODE"
      00012258        52
      00012260        41
      00012268        50
      00012270        48
      00012278        4D
      00012280        4F
      00012288        44
      00012290        45
A1468 000122A0                          .even
A1469 000122A0 000122C0'                .field graphmode
A1470 000122C0      0D5F  graphmode:    calla docreate
      000122D0 000010A0'
A1471 000122F0 00000000  fgraphmode:    .int 0                      ;fill mode
A1472                       *
A1473                       * - DRAWMODE -
A1474                       *      Poe o sistema  em modo de desenho DRAW.
A1475                       *
A1476 00012310 00012220' h_drawmode:    .int h_graphmode
A1477 00012330        08                .byte 8,0
```

```
      00012338        00
A1478 00012340        44                    .string "DRAWMODE"
      00012348        52
      00012350        41
      00012358        57
      00012360        4D
      00012368        4F
      00012370        44
      00012378        45
A1479 00012380                              .even
A1480 00012380 000123A0'                    .field drawmode
A1481 000123A0      09E0  drawmode:         movi true, a0
      000123B0 FFFFFFFF
A1482 000123D0      0580                    move a0,@fgraphmode
      000123E0 000122F0'
A1483 00012400      0960                    rets
A1484                     *
A1485                     * - FILLMODE -
A1486                     *       Poe o sistema  em modo de desenho DRAW.
A1487                     *
A1488 00012410 00012310' h_fillmode:        .int h_drawmode
A1489 00012430        08                    .byte 8,0
      00012438        00
A1490 00012440        46                    .string "FILLMODE"
      00012448        49
      00012450        4C
      00012458        4C
      00012460        4D
      00012468        4F
      00012470        44
      00012478        45
A1491 00012480                              .even
A1492 00012480 000124A0'                    .field fillmode
A1493 000124A0      09C0  fillmode:         movi false, a0
      000124B0      0000
A1494 000124C0      0580                    move a0,@fgraphmode
      000124D0 000122F0'
A1495 000124F0      0960                    rets
A1496                     *
A1497                     * y x ALINETO -
A1498                     *       Traca linha, XY absoluto.
A1499                     *       Drawmode - usa Brush
A1500                     *       Fillmode - usa Boundary
A1501                     *
A1502 00012500 00012410' h_alineto:         .int h_fillmode
A1503 00012520        07                    .byte 07,0
      00012528        00
A1504 00012530        41                    .string "ALINETO"
      00012538        4C
      00012540        49
      00012548        4E
      00012550        45
      00012558        54
      00012560        4F
A1505 00012570                              .even
A1506 00012570 00012590'                    .field alineto
A1507 00012590      05A0  alineto:          move @fgraphmode,a0
      000125A0 000122F0'
A1508 000125C0      0B40                    cmpi false,a0
      000125D0      FFFF
A1509 000125E0      CA04                    jrz alineto1
A1510 000125F0      0D5F                    calla lineb
      00012600 0000F110'
A1511 00012620      C003                    jruc alineto2
A1512 00012630      0D5F  alineto1:         calla linef
      00012640 000107D0'
A1513 00012660      0960  alineto2:         rets
A1514                     *
A1515                     * y x RLINETO -
A1516                     *       Traca linha X Y , relativo ao ponto corrente do Stencil.
A1517                     *
A1518 00012670 00012500' h_rlineto:         .int h_alineto
A1519 00012690        07                    .byte 07,0
      00012698        00
A1520 000126A0        52                    .string "RLINETO"
      000126A8        4C
```

```
      000126B0        49
      000126B8        4E
      000126C0        45
      000126C8        54
      000126D0        4F
A1521 000126E0                          .even
A1522 000126E0 00012700'                .field rlineto
A1523 00012700     0D5F  rlineto:       calla ddup
      00012710 00011810'
A1524 00012730     0D5F                 calla vcpt
      00012740 000119B0'
A1525 00012760     0D5F                 calla doublestore
      00012770 000118E0'
A1526 00012790     0D5F                 calla scale
      000127A0 00011E60'
A1527 000127C0     0D5F                 calla shear
      000127D0 00012060'
A1528 000127F0     0D5F                 calla rrotation
      00012800 00011180'
A1529 00012820     0D5F                 calla curstencil
      00012830 0000EF60'
A1530 00012850     0D5F                 calla fetch
      00012860 00000C10'
A1531 00012880     0D5F                 calla getcpt
      00012890 0000EC50'
A1532 000128B0     9560                 move *pstk+,a0                ;x
A1533 000128C0     9561                 move *pstk+,a1                ;y
A1534 000128D0     9562                 move *pstk+,a2                ;x
A1535 000128E0     9563                 move *pstk+,a3                ;y
A1536 000128F0     4040                 add a2,a0
A1537 00012900     4061                 add a3,a1
A1538 00012910     A02B                 move a1,-*pstk
A1539 00012920     A00B                 move a0,-*pstk
A1540 00012930     0D5F                 calla alineto
      00012940 00012590'
A1541 00012960     0960                 rets
A1542                       *
A1543                       * - ORIGIN adr
A1544                       *      Variavel dupla  que  contem o ponto XY base  para os
A1545                       *      lineto  seguintes:
A1546                       *
A1547 00012970 00012670' h_origin:      .int h_rlineto
A1548 00012990       06                 .byte 06,0
      00012998       00
A1549 000129A0       4F                 .string "ORIGEN"
      000129A8       52
      000129B0       49
      000129B8       47
      000129C0       49
      000129C8       4E
A1550 000129D0                          .even
A1551 000129D0 000129F0'                .field  origin
A1552 000129F0     0D5F  origin:        calla rdoublevar
      00012A00 000116B0'
A1553 00012A20 00000000                 .int 0
A1554 00012A40 00000000                 .int 0
A1555                       *
A1556                       * y x y x ADDXY y x
A1557                       *      Funcao auxiliar, soma dois pontos XY:
A1558                       *      Y1 X1  +  Y2 X2  ->  Y1+Y2     X1+X2
A1559                       *
A1560 00012A60 00012970' h_addxy:       .int h_origin
A1561 00012A80       05                 .byte 05,0
      00012A88       00
A1562 00012A90       41                 .string "ADDXY"
      00012A98       44
      00012AA0       44
      00012AA8       58
      00012AB0       59
A1563 00012AC0                          .even
A1564 00012AC0 00012AE0'                .field addxy
A1565 00012AE0     9560  addxy:         move *pstk+,a0
A1566 00012AF0     9561                 move *pstk+,a1
A1567 00012B00     9562                 move *pstk+,a2
A1568 00012B10     9563                 move *pstk+,a3
A1569 00012B20     4040                 add a2,a0
```

```
A1570 00012B30      4061                    add a3,a1
A1571 00012B40      A02B                    move a1,-*pstk
A1572 00012B50      A00B                    move a0,-*pstk
A1573 00012B60      0960                    rets
A1574                         *
A1575                         * y x LINETO -
A1576                         *       Traca linha do ponto corrente ate Origin + XY.
A1577                         *
A1578 00012B70 00012A60' h_lineto:          .int h_addxy
A1579 00012B90       06                     .byte 06,0
      00012B98       00
A1580 00012BA0       4C                     .string "LINETO"
      00012BA8       49
      00012BB0       4E
      00012BB8       45
      00012BC0       54
      00012BC8       4F
A1581 00012BD0                              .even
A1582 00012BD0 00012BF0'                    .field lineto
A1583 00012BF0      0D5F  lineto:           calla ddup
      00012C00 00011810'
A1584 00012C20      0D5F                    calla vcpt
      00012C30 000119B0'
A1585 00012C50      0D5F                    calla doublestore
      00012C60 000118E0'
A1586 00012C80      0D5F                    calla scale
      00012C90 00011E60'
A1587 00012CB0      0D5F                    calla shear
      00012CC0 00012060'
A1588 00012CE0      0D5F                    calla rrotation
      00012CF0 00011180'
A1589 00012D10      0D5F                    calla origin
      00012D20 000129F0'
A1590 00012D40      0D5F                    calla doublefetch
      00012D50 00011740'
A1591 00012D70      0D5F                    calla addxy
      00012D80 00012AE0'
A1592 00012DA0      05A0                    move @fgraphmode,a0
      00012DB0 000122F0'
A1593 00012DD0      0B40                    cmpi false,a0
      00012DE0      FFFF
A1594 00012DF0      CA04                    jrz linetol
A1595 00012E00      0D5F                    calla lineb
      00012E10 0000F110'
A1596 00012E30      C003                    jruc lineto2
A1597 00012E40      0D5F  linetol:          calla linef
      00012E50 000107D0'
A1598 00012E70      0960  lineto2:          rets
A1599                         *
A1600                         * y x AMOVETO -
A1601                         *       Atualiza Current Point do Stencil corrente.
A1602                         *       Y X ->  CPT
A1603                         *
A1604 00012E80 00012B70' h_amoveto:         .int h_lineto
A1605 00012EA0       07                     .byte 07,
      00012EA8       00
A1606 00012EB0       41                     .string "AMOVETO"
      00012EB8       4D
      00012EC0       4F
      00012EC8       56
      00012ED0       45
      00012ED8       54
      00012EE0       4F
A1607 00012EF0                              .even
A1608 00012EF0 00012F10'                    .field amoveto
A1609 00012F10      0D5F  amoveto:          calla curstencil
      00012F20 0000EF60'
A1610 00012F40      0D5F                    calla fetch
      00012F50 00000C10'
A1611 00012F70      0D5F                    calla setcpt
      00012F80 0000E0D0'
A1612 00012FA0      0960                    rets
A1613                         *
A1614                         * y x RMOVETO -
A1615                         *       Atualiza Current Point do Stencil corrente.
A1616                         *       CPT = CPT + YX
```

```
A1617                          *
A1618 00012FB0 00012E80' h_rmoveto:      .int h_amoveto
A1619 00012FD0       07                  .byte 07,0
      00012FD8       00
A1620 00012FE0       52                  .string "RMOVETO"
      00012FE8       4D
      00012FF0       4F
      00012FF8       56
      00013000       45
      00013008       54
      00013010       4F
A1621 00013020                           .even
A1622 00013020 00013040'                 .field rmoveto
A1623 00013040      0D5F  rmoveto:       calla ddup
      00013050 00011810'
A1624 00013070      0D5F                 calla vcpt
      00013080 000119B0'
A1625 000130A0      0D5F                 calla doublestore
      000130B0 000118E0'
A1626 000130D0      0D5F                 calla scale
      000130E0 00011E60'
A1627 00013100      0D5F                 calla shear
      00013110 000120'0'
A1628 00013130      0D5F                 calla rrotation
      00013140 00011180'
A1629 00013160      0D5F                 calla curstencil
      00013170 0000EF60'
A1630 00013190      0D5F                 calla fetch
      000131A0 00000C10'
A1631 000131C0      0D5F                 calla getcpt
      000131D0 0000EC50'
A1632 000131F0      0D5F                 calla addxy
      00013200 00012AE0'
A1633 00013220      0D5F                 calla curstencil
      00013230 0000EF60'
A1634 00013250      0D5F                 calla fetch
      00013260 00000C10'
A1635 00013280      0D5F                 calla setcpt
      00013290 0000EDD0'
A1636 000132B0      0960                 rets
A1637                          *
A1638                          * y x MOVETO -
A1639                          *
A1640                          *       Atualiza CPT do Stencil corrente.
A1641                          *       ORIGIN + XY -> CPT
A1642                          *
A1643 000132C0 00012FB0' h_moveto:       .int h_rmoveto
A1644 000132E0       06                  .byte 06,0
      000132E8       00
A1645 000132F0       4D                  .string "MOVETO"
      000132F8       4F
      00013300       56
      00013308       45
      00013310       54
      00013318       4F
A1646 00013320                           .even
A1647 00013320 00013340'                 .field moveto
A1648 00013340      0D5F  moveto:        calla ddup
      00013350 00011810'
A1649 00013370      0D5F                 calla vcpt
      00013380 000119B0'
A1650 000133A0      0D5F                 calla doublestore
      000133B0 000118E0'
A1651 000133D0      0D5F                 calla scale
      000133E0 00011E60'
A1652 00013400      0D5F                 calla shear
      00013410 00012D60'
A1653 00013430      0D5F                 calla rrotation
      00013440 00011180'
A1654 00013460      0D5F                 calla origin
      00013470 000129F0'
A1655 00013490      0D5F                 calla doublefetch
      000134A0 00011740'
A1656 000134C0      0D5F                 calla addxy
      000134D0 00012AE0'
A1657 000134F0      0D5F                 calla curstencil
```

```
      00013500 0000EF60'
A1658 00013520     0D5F                 calla fetch
      00013530 00000C10'
A1659 00013550     0D5F                 calla setcpt
      00013560 0000EDD0'
A1660 00013580     0960                 rets
A1661                      *
A1662                      *
A1663                      * dim  ARRAY name
A1664                      *
A1665                      *            Cria a matriz  unidimensional name com dimensao DIM.
A1666                      *
A1667 00013590 000132C0' h_array:       .int h_moveto
A1668 000135B0       05                 .byte 05,0
      000135B8       00
A1669 000135C0       41                 .string "ARRAY"  .
      000135C8       52
      000135D0       52
      000135D8       41
      000135E0       59
A1670 000135F0                          .even
A1671 000135F0 00013610'                .field array
A1672 00013610     0D5F  array:         calla builds
      00013620 00003EE0'
A1673 00013640     9560                 move *pstk+,a0                ;endereco para dois
A1674 00013650     9561                 move *pstk+,a1                ;DIM
A1675 00013660     05A3                 move @fdp,a3
      00013670 00001810'
A1676 00013690     5684                 clr a4
A1677 000136A0     9083  arrayl:       move a4,*a3+
A1678 000136B0     3C41                 dsj a1,arrayl
A1679 000136C0     0583                 move a3,@fdp
      000136D0 00001810'
A1680 000136F0     A00B                 move a0,-*pstk                ;
A1681 00013700     0D5F                 calla does
      00013710 000040D0'
A1682 00013730     95E0  doarray:       move *sp+,a0
A1683 00013740     9561                 move *pstk+,a1
A1684 00013750     20A1                 sla 5,a1                      ;*32
A1685 00013760     4020                 add a1,a0
A1686 00013770     A00B                 move a0,-*pstk
A1687 00013780     0960                 rets
A1688                      *
A1689                      *  ( texto )
A1690                      *
A1691                      *No modo  interpretativo coloca-se o string seguinte (ate ")"
A1692                      *no Word Buffer e deixa na pilha o endereco e o comprimento.
A1693                      *No  modo Compilativo coloca-se o string seguinte no  Dicionario,
A1694                      *Na sua execusao coloca o endereco e o comprimento na pilha
A1695                      *
A1696 00013790 00013590' h_leftpar:     .int h_array
A1697 000137B0       01                 .byte 01,01                   ;immediate
      000137B8       01
A1698 000137C0       28                 .string "("
A1699 000137D0                          .even
A1700 000137D0 000137F0'                .field leftpar
A1701 000137F0     05A0  leftpar:       move @fmode,a0
      00013800 00003AC0'
A1702 00013820     0B40                 cmpi 0,a0
      00013830     FFFF
A1703 00013840     CB12                 jrnz leftparl
A1704 00013850     09C0                 movi 29h,a0,1
      00013860     0029
A1705 00013870     A00B                 move a0,-*pstk
A1706 00013880     0D5F                 calla token          ;poe string na word buffer
      00013890 00002FA0'
A1707 000138B0     05A3                 move @fdp,a3
      000138C0 00001810'
A1708 000138E0     1003                 addk 32,a3
A1709 000138F0     8462                 move *a3,a2
A1710 00013900     0B82                 andi 0ffh,a2                  ;nchar
      00013910 FFFFFF00
A1711 00013930     1203                 addk 16,a3
A1712 00013940     A06B                 move a3,-*pstk                ;endereco
A1713 00013950     A04B                 move a2,-*pstk                ;nchar
A1714 00013960     C024                 jruc leftpar3
```

```
  0D5F  leftparl:      calla compile
01C00'
  0D5F                 calla rquote
004730'
  05A2                 move @fdp,a2
001810'
  1402                 subk 32,a2
  0582                 move a2,@fdp
001810'
  09C0                 movi 29h,a0,l
  0029
  A00B                 move a0,-*pstk
  0D5F                 calla token                    ;atualiza @fdp
002FA0'
  05A0                 move @fdp,a0
001810'
  1000                 addk 32,a0
  8401                 move *a0, a1
  0B81                 andi 0ffh,a1
FFFF00
  1200                 addk 16,a0
  1FE1                 btst 0,a1
  CA01                 jrz  leftpar2
  1021                 addk 1,a1
  2461  leftpar2:      sll 3,a1
  4020                 add a1,a0
  0580                 move a0,@fdp
001810'
  0960  leftpar3:      rets
        *
        * - CURBMFONT adr
        *       Variavel de controle. Contem o endereco  da matriz fonte de
        *       caracteres em BitMap.
        *
013790' h_curbmfont:   .int h_leftpar
    08                 .byte 08,0
    00
    43                 .string "CURBMFONT"
    55
    52
    42
    4D
    46
    4F
    4E
    54
                       .even
013C60'                .field curbmfont
  0D5F  curbmfont:     calla docreate
00010A0'
0000000                .int 0
        *
        * - ADDRSHOW adr
        *       Variavel de controle que contem o  endereco  do  string
        *       sendo utilizado em  BMSHOW e TSHOW.
        *
0013BC0' h_addrshow:   .int h_curbmfont
    08                 .byte 08h,0h
    00
    41                 .string "ADDRSHOW"
    44
    44
    52
    53
    48
    4F
    57
                       .even
0013D40'               .field addrshow
  0D5F  addrshow:      calla docreate
00010A0'
0000000                .int 0
        *
        * - ADJUSTXY adr
        *       Variavel dupla que contem o incremento DeltaX Delta Y
        *       que serao usados entre cada caracter desenhado por BSHOW.
```

```
A1763                          *
A1764 00013D90 00013CB0' h_adjustxy:     .int h_addrshow
A1765 00013D80       08                  .byte 08,0
      00013D88       00
A1766 00013DC0       41                  .string "ADJUSTXY"
      00013DC8       44
      00013DD0       4A
      00013DD8       55
      00013DE0       53
      00013DE8       54
      00013DF0       58
      00013DF8       59
A1767 00013E00                           .even
A1768 00013E00 00013E20'                 .field adjustxy
A1769 00013E20     0D5F  adjustxy:       calla rdoublevar
      00013E30 000116B0'
A1770 00013E50 00000000                  .int 0
A1771 00013E70 00000000                  .int 0
A1772                          *
A1773                          * adr BFETCH b
A1774                          *      Funcao auxiliar, pega byte LSB do endereco.
A1775                          *
A1776 00013E90 00013D90' h_bfetch:       .int h_adjustxy
A1777 00013EB0       02                  .byte 02h,0
      00013EB8       00
A1778 00013EC0       42                  .string "B@"
      00013EC8       40
A1779 00013ED0                           .even
A1780 00013ED0 00013EF0'                 .field bfetch
A1781 00013EF0     9560  bfetch:         move *pstk+,a0
A1782 00013F00     8401                  move *a0,a1
A1783 00013F10     0B81                  andi 0ffh,a1
      00013F20 FFFFFF00
A1784 00013F40     A00B                  move a0,-*pstk
A1785 00013F50     0960                  rets
A1786                          *
A1787                          * adr n BSHOW -
A1788                          *      BitMap SHOW - pega o string de "adr" com "n" caracteres e ,
A1789                          *      usando Curbmfont, escreve em Curstencil.
A1790                          *
A1791 00013F60 00013E90' h_bshow:        .int h_bfetch
A1792 00013F80       05                  .byte 05,0
      00013F88       00
A1793 00013F90       42                  .string "BSHOW"
      00013F98       53
      00013FA0       48
      00013FA8       4F
      00013FB0       57
A1794 00013FC0                           .even
A1795 00013FC0 00013FE0'                 .field bshow
A1796 00013FE0     0D5F  bshow:          calla swap
      00013FF0 000001E0'
A1797 00014010     0D5F                  calla addrshow
      00014020 00013D40'
A1798 00014040     0D5F                  calla store
      00014050 00000CB0'
A1799 00014070     0D5F                  calla lit
      00014080 00001230'
A1800 000140A0 00000000                  .int 0
A1801 000140C0     0D5F                  calla rdo
      000140D0 00001D50'
A1802 000140F0 00014480'                 .field bshow2
A1803 00014110     0D5F  bshow1:         calla i
      00014120 00002080'
A1804 00014140     0D5F                  calla lit
      00014150 00001230'
A1805 00014170 00000008                  .int 8
A1806 00014190     0D5F                  calla times
      000141A0 00004E80'
A1807 000141C0     0D5F                  calla addrshow
      000141D0 00013D40'
A1808 000141F0     0D5F                  calla fetch
      00014200 00000C10'
A1809 00014220     0D5F                  calla plus
      00014230 00002890'
A1810 00014250     0D5F                  calla bfetch
```

```
        00014260 00013EF0'
A1811   00014280      0D5F                    calla curbmfont
        00014290 00013C60'
A1812   000142B0      0D5F                    calla fetch
        000142C0 00000C10'
A1813   000142E0      0D5F                    calla execute
        000142F0 00000FE0'
A1814   00014310      0D5F                    calla curstencil
        00014320 0000EF60'
A1815   00014340      0D5F                    calla fetch
        00014350 00000C10'
A1816   00014370      0D5F                    calla paste
        00014380 0000E0A0'
A1817   000143A0      0D5F                    calla adjustxy
        00014380 00013E20'
A1818   000143D0      0D5F                    calla doublefetch
        000143E0 00011740'
A1819   00014400      0D5F                    calla rmoveto
        00014410 00013040'
A1820   00014430      0D5F                    calla rloop
        00014440 00001E70'
A1821   00014460 00014110'                    .field bshow1
A1822   00014480      0960  bshow2:           rets
A1823                       *
A1824                       * - CURCPT cpy cpx
A1825                       *       Retorno o Current  point do Stencil corrente, poe na pilha.
A1826                       *
A1827   00014490 00013F60'  h_curcpt:         .int h_bshow
A1828   000144B0        06                    .byte 06,0
        000144B8        00
A1829   000144C0        43                    .string "CURCPT"
        000144C8        55
        000144D0        52
        000144D8        43
        000144E0        50
        000144E8        54
A1830   000144F0                              .even
A1831   000144F0 00014510'                    .field curcpt
A1832   00014510      0D5F  curcpt:                   calla curstencil
        00014520 0000EF60'
A1833   00014540      0D5F                    calla fetch
        00014550 00000C10'
A1834   00014570      0D5F                    calla getcpt
        00014580 0000EC50'
A1835   000145A0      0960                    rets
A1836                       *
A1837                       * - CURTFONT a
A1838                       *       Variavel de controle que contem o endereco de execucao
A1839                       *       da matriz de codificacao de um Fonte de trajetorias,
A1840                       *       e sera usado por TSHOW.
A1841                       *
A1842   000145B0 00014490'  h_curtfont        .int h_curcpt
A1843   000145D0        08                    .byte 08,0
        000145D8        00
A1844   000145E0        43                    .string "CURTFONT"
        000145E8        55
        000145F0        52
        000145F8        54
        00014600        46
        00014608        4F
        00014610        4E
        00014618        54
A1845   00014620                              .even
A1846   00014620 00014640'                    .field curtfont
A1847   00014640      0D5F  curtfont:         calla docreate
        00014650 000010A0'
A1848   00014670 00000000                     .int 0
A1849                       *
A1850                       * SELECTBMFONT array
A1851                       *       Coloca o endereco de Array em CURBMFONT
A1852                       *
A1853   00014690 000145B0'  h_selectbmfont:   .int h_curtfont
A1854   000146B0        0C                    .byte 12,01              ;immediate
        000146B8        01
A1855   000146C0        53                    .string "SELECTBMFONT"
        000146C8        45
```

```
        000146D0        4C
        000146D8        45
        000146E0        43
        000146E8        54
        000146F0        42
        000146F8        4D
        00014700        46
        00014708        4F
        00014710        4E
        00014718        54
A1856   00014720                        .even
A1857   00014720 00014740'              .field selectbmfont
A1858   00014740      05A0  selectbmfont:  move @fmode,a0          ;testa se modo compila-
        00014750 00003AC0'
A1859                       *                                        tivo
A1860   00014770      0B40              cmpi 0,a0
        00014780      FFFF
A1861   00014790      CB0A               jrnz selectbmfont1
A1862   000147A0      0D5F              calla tick
        000147B0 0000AA20'
A1863   000147D0      0D5F              calla curbmfont
        000147E0 00013C60'
A1864   00014800      0D5F              calla store
        00014810 00000CB0'
A1865   00014830      C00F              jruc selectbmfont2
A1866   00014840      0D5F  selectbmfont1: calla bracktick        ;compilative mode
        00014850 0000AC70'
A1867   00014870      0D5F              calla compile             ;pega o endereco
        00014880 00001C00'
A1868   000148A0      0D5F              calla curbmfont          ;compila carga de curbmfont
        000148B0 00013C60'
A1869   000148D0      0D5F              calla compile
        000148E0 00001C00'
A1870   00014900      0D5F              calla store
        00014910 00000CB0'
A1871   00014930      0960  selectbmfont2: rets
A1872                       *
A1873                       * true x x .... DROPTRUE -
A1874                       *             "drop until true"
A1875                       *
A1876   00014940 00014690' h_droptrue:    .int h_selectbmfont
A1877   00014960        08                .byte 08,0
        00014968        00
A1878   00014970        44                .string "DROPTRUE"
        00014978        52
        00014980        4F
        00014988        50
        00014990        54
        00014998        52
        000149A0        55
        000149A8        45
A1879   000149B0                        .even
A1880   000149B0 000149D0'               .field droptrue
A1881   000149D0      05A1  droptrue:     move @spo,a1
        000149E0 00009570'
A1882   00014A00      9560  droptrue1:   movo *pstk+,a0
A1883   00014A10      0B40              cmpi true,a0
        00014A20      0000
A1884   00014A30      CA02              jrz droptruesai
A1885   00014A40      4961              cmp pstk,a1
A1886   00014A50      C3FA              jrhi droptrue1
A1887   00014A60      0960  droptruesai:  rets
A1888                       *
A1889                       * SELECTTFONT array
A1890                       *       Coloca o endereco do Array em CURTFONT.
A1891                       *
A1892   00014A70 00014940' h_selecttfont: .int h_droptrue
A1893   00014A90        0B                .byte 11,01              ;immediate
        00014A98        01
A1894   00014AA0        53                .string "SELECTTFONT"
        00014AA8        45
        00014AB0        4C
        00014AB8        45
        00014AC0        43
        00014AC8        54
        00014AD0        54
```

```
      00014AD8        46
      00014AE0        4F
      00014AE8        4E
      00014AF0        54
A1895 00014B00                          .even
A1896 00014B00 00014B20'                .field selecttfont
A1897 00014B20      05A0  selecttfont:  move @fmode,a0          ;testes se modo compilativo
      00014B30 00003AC0'
A1898 00014B50      0B40                cmpi 0,a0
      00014B60      FFFF
A1899 00014B70      CB0A                jrnz selecttfont1
A1900 00014B80      0D5F                calla tick
      00014B90 0000AA20'
A1901 00014BB0      0D5F                calla curtfont
      00014BC0 00014640'
A1902 00014BE0      0D5F                calla store
      00014BF0 00000CB0'
A1903 00014C10      C00F                jruc selecttfont2
A1904 00014C20      0D5F  selecttfont1: calla bracktick         ;compilative mode
      00014C30 0000AC70'
A1905 00014C50      0D5F                calla compile           ;pega o endereco
      00014C60 00001C00'
A1906 00014C80      0D5F                calla curtfont:         ;compila carga de curtfont
      00014C90 00014640'
A1907 00014CB0      0D5F                calla compile
      00014CC0 00001C00'
A1908 00014CE0      0D5F                calla store
      00014CF0 00000CB0'
A1909 00014D10      0960  selecttfont2: rets
A1910                     *
A1911                     * TSHOW
A1912                     *       adr  n  Tshow
A1913                     *       Trajectory Show -> pega o String de endereco adr,
A1914                     *       com n caracteres e o desenho em CURSTENCIL, usando CURTFONT.
A1915                     *       Se em DRAWMODE, usa CURBRUSH.
A1916                     *       Se em FILLMODE, usa BOUNDARY e FILL.
A1917                     *       End  n  Tshow
A1918                     *       Swap  n  Tshow
A1919                     *       Swap addrshow!
A1920                     *       True Swap
A1921                     *       0 DO
A1922                     *       I 8 * addrshow @ + b@
A1923                     *       Curtfont @
A1924                     *       CURCPT ORIGIN D!
A1925                     *       Execute
A1926                     *       Adjust de RMOVETO
A1927                     *       LOOP
A1928                     *       GRAPHMODE @ IF DROPTRUE ELSE FILL THEN
A1929                     *
A1930                     *
A1931 00014D20 00014A70' h_tshow:       .int h_selecttfont
A1932 00014D40        05                .byte 05,0
      00014D48        00
A1933 00014D50        54                .string "TSHOW"
      00014D58        53
      00014D60        48
      00014D68        4F
      00014D70        57
A1934 00014D80                          .even
A1935 00014D80 00014DA0'                .field tshow
A1936 00014DA0      0D5F  tshow:        calla swap
      00014DB0 000001E0'
A1937 00014DD0      0D5F                calla addrshow
      00014DE0 00013D40'
A1938 00014E00      0D5F                calla store
      00014E10 00000CB0'
A1939 00014E30      0D5F                calla lit
      00014E40 00001230'
A1940 00014E60 FFFFFFFF                 .int true
A1941 00014E80      0D5F                calla swap
      00014E90 000001E0'
A1942 00014EB0      0D5F                calla lit
      00014EC0 00001230'
A1943 00014EE0 00000000                 .int 0
A1944 00014F00      0D5F                calla rdo
      00014F10 00001D50'
```

```
A1945 00014F30 000152C0'                    .field tshow2
A1946 00014F50      0D5F  tshow1:                 calla i
      00014F60 00002080'
A1947 00014F80      0D5F                    calla lit
      00014F90 00001230'
A1948 00014FB0 00000008                     .int 8
A1949 00014FD0      0D5F                    calla times
      00014FE0 00004E80'
A1950 00015000      0D5F                    calla addrshow
      00015010 00013D40'
A1951 00015030      0D5F                    calla fetch
      00015040 00000C10'
A1952 00015060      0D5F                    calla plus
      00015070 00002890'
A1953 00015090      0D5F                    calla bfetch
      000150A0 00013EF0'
A1954 000150C0      0D5F                    calla curtfont
      000150D0 00014640'
A1955 000150F0      0D5F                    calla fetch
      00015100 00000C10'
A1956 00015120      0D5F                    calla curcpt
      00015130 00014510'
A1957 00015150      0D5F                    calla origin
      00015160 000199F0'
A1958 00015180      0D5F                    calla doublestore
      00015190 000118E0'
A1959 000151B0      0D5F                    calla execute
      000151C0 00000FE0'
A1960 000151E0      0D5F                    calla adjustxy
      000151F0 00013E20'
A1961 00015210      0D5F                    calla doublefetch
      00015220 00011740'
A1962 00015240      0D5F                    calla rmoveto
      00015250 00013040'
A1963 00015270      0D5F                    calla rloop
      00015280 00001E70'
A1964 000152A0 00014F50'                    .field tshow1
A1965 000152C0      0D5F  tshow2:                 calla graphmode
      000152D0 000122C0'
A1966 000152F0      0D5F                    calla fetch
      00015300 00000C10'
A1967 00015320      0D5F                    calla cbranch
      00015330 00001600'
A1968 00015350 000153F0'                    .field tshow3
A1969 00015370      0D5F                    calla droptrue
      00015380 000149D0'
A1970 000153A0      0D5F                    calla branch
      000153B0 00001530'
A1971 000153D0 00015420'                    .field tshow4
A1972 000153F0      0D5F  tshow3:                 calla fill
      00015400 000105B0'
A1973 00015420      0960  tshow4:                 rets
A1974
 2314 00015430      COFF  endp:            jruc endp
 2315                                      .end

No Errors, No Warnings
```

LABEL	VALUE	DEFN	REF					
ABORTQUOTE	00009850'	2034	2033					
ABS	000054B0'	1280	1279	1796				
ADDRSHOW	00013D40'	A1757	A1756	A1797	A1807	A1937	A1950	
ADDXY	00012AE0'	A1565	A1564	A1591	A1632	A1656		
ADJUSTXY	00013E20'	A1769	A1768	A1817	A1960			
AGAIN	00006000'	1439	1438					
ALINETO	00012590'	A1507	A1506	A1540				
ALINETO1	00012630'	A1512	A1509					
ALINETO2	00012660'	A1513	A1511					
ALLOT	000018B0'	483	482	972				
AMOVETO	00012F10'	A1609	A1608					
AND	000004E0'	143	142	1619	2171			
ARRAY	00013610'	A1672	A1671					
ARRAY1	000136A0'	A1677	A1678					
ASCII	00007050'	1611	1610					
ASTK	0000000C	30	555	557	558	571	572	
			576	577	579	603	615	
			626	637	638	1982		
BASE	00006B50'	1547	1546	1720				
BEGIN	00005EF0'	1420	1419					
BFETCH	00013EF0'	A1781	A1780	A1810	A1953			
BLACK	0000C530'	A 202	A 201					
BMARK	00005A40'	1361	1360	1420				
BOUNDARY	0000FB20'	A 936	A 935					
BRACKCOMPILE	0000AB90'	2202	2201					
BRACKTICK	0000AC70'	2213	2212	A1866	A1904			
BRANCH	00001530'	428	427	578	1397	1440	1465	
			2243	2269	2272	2304	2309	
			2311	A1970				
BRESOLVE	00005B10'	1372	1371	1432	1441	1466	1496	
			1513					
BSHOW	00013FE0'	A1796	A1795					
BSHOW1	00014110'	A1803	A1821					
BSHOW2	00014480'	A1822	A1802					
BUILDS	00003EE0'	965	964	1887	A 447	A1273	A1672	
CBRANCH	00001600'	442	441	1384	1431	1452	1728	
			1748	1783	2115	2119	2159	
			2172	2240	2259	2263	2295	
			2301	A1967				
CBRANCHSAI	000016A0'	451	448					
CCOMMA	00001AA0'	511	510	950	968	2267		
CFETCH	00000D50'	299	298					
CLEAVE	00006A70'	1535	1534					
CMISSING	0000A000'	2114	2113	2192	2247	2277		
CMISSING1	0000A310'	2131	2120					
CMISSING2	0000A480'	2140	2116					
COLON	00003840'	930	929					
COLOR0	00000018	A 48						
COLOR1	00000019	A 49	A1114					
COMMA	000019B0'	497	496	867	895	908	1337	
			1372	1840	1889	2203	2268	
			2280					
COMMAQUOTE	00004890'	1098	1097	1149	1162			
COMMAQUOTE1	00004A20'	1112	1110					
COMP1	0000B390'	2274	2260					
COMP2	0000B310'	2271	2264					
COMP3	0000B340'	2272	2270					
COMP4	0000B4E0'	2281	2273					
COMPILATOR	0000B0B0'	2257	2256	2303				
COMPILE	00001C00'	529	528	893	906	1147	1160	
			1383	1396	1430	1439	1451	
			1464	1477	1490	1507	1524	
			1535	1620	1838	2034	2278	
			A1715	A1867	A1869	A1905	A1907	
COMPILER	00002A20'	744	743	783	787	2176		
COMPILERWORD	00002CA0'	776	775					
COMPOSE	0000DF10'	A 522	A 521					
CONSTANT	000037F0'	892	891					
CONTEXT	000025E0'	684	683	932	1893	2153		
CONTROL	C0000080	A 14	A 129	A 132	A 151	A 154	A 830	
			A 833	A 923	A 962	A 965	A1053	
			A1109	A1112	A1174			
CONVDP	C0000140	A 18	A 564	A 607	A 666	A 829	A 961	
			A1108					

LABEL	VALUE	DEFN	REF					
CONVSP	C0000130	A 15	A 577	A 620	A 665	A 815	A 960	
CORE	00002AE0'	754	753					
COUNT	0000001A	A 51	A1157	A1158				
CREATE	00003400'	857	856	892	905	934	965	
CREATE2	000036A0'	873	874					
CREATE3	000036F0'	878	876					
CSTACK	00009C00'	2087	2086	2297				
CSTORE	00000E20'	312	311	1690				
CURBMFONT	00013C60'	A1745	A1744	A1811	A1863	A1868		
CURBRUSH	0000F040'	A 781	A 780					
CURCPT	00014510'	A1832	A1831	A1956				
CURRENT	000026C0'	694	693	705	780	864	930	
			1895	2121	2128	2163		
CURSTENCIL	0000EF60'	A 769	A 768	A1529	A1609	A1629	A1633	
			A1657	A1814	A1832			
CURTFONT	00014640'	A1847	A1846	A1901	A1906	A1954		
DADDR	00000012	A 42	A 713	A 823	A 893	A 898	A 902	
			A 913	A1100				
DDUP	00011810'	A1308	A1307	A1523	A1583	A1623	A1648	
DEFINED	0000A520'	2152	2151	2191	2298			
DEFINED2	0000A9B0'	2180	2160	2173				
DETACH	0000E360'	A 596	A 595					
DIFO	0000AD50'	2224	2223	2239	2258			
DIFO2	0000ADA0'	2229	2227					
DIP	000017E0'	473	472	1613	2095	2125		
DIV	00004F60'	1188	1187					
DIVMOD	00005140'	1220	1219	1722				
DIVMOD1	000051C0'	1228	1226					
DO	00006470'	1477	1476					
DOARRAY	00013730'	A1682						
DOCONSTANT	00001170'	366	365	894	1995			
DOCREATE	000010A0'	353	352	473	684	694	744	
			754	765	907	919	1547	
			1632	1643	1653	1664	1674	
			1969	A 80	A 93	A 106	A 481	
			A 769	A 781	A 936	A1190	A1470	
			A1745	A1757	A1847			
DOES	000040D0'	984	983	1890	A 467	A1280	A1681	
DOSTENCIL	0000D950'	A 447	A 446	A 509				
DOT	000085A0'	1827	1622	1826				
DOTQUOTE	00004C80'	1147	1146	2036				
DOUBLEFETCH	00011740'	A1293	A1292	A1590	A1655	A1818	A1961	
DOUBLESTORE	000118E0'	A1325	A1324	A1525	A1585	A1625	A1650	
			A1958					
DOUBLEVAR	000115A0'	A1273	A1272					
DOVOC	00008A90'	1891	A 67					
DPTCH	00000013	A 43	A 657	A 692	A 704	A 712	A 825	
			A 956	A1104				
DPYADR	C00001E0	A 25						
DPYCTL	C0000080	A 26						
DPYINT	C00000A0	A 27						
DPYSTRT	C0000090	A 28						
DPYTAP	C00001B0	A 29						
DRAWMODE	000123A0'	A1481	A1480					
DROP	00000150'	82	81	1763	2161	2174	2245	
			2274	2275				
DROPTRUE	000149D0'	A1881	A1880	A1969				
DROPTRUE1	00014A00'	A1882	A1886					
DROPTRUESAI	00014A60'	A1887	A1884					
DUP	000000A0'	70	69	777	778	861	1492	
			1509	1705	1780	1795	1892	
			2124	2157	2167	2257		
DYDX	00000017	A 47	A 660	A 711	A 807	A1163		
EINT1	FFFFFFC0	20						
ELSE	00005CC0'	1396	1395					
ENDP	00015430'	2314	474	1980	1981	1982	1983	
			1996	2314				
ENTRY	00002790'	705	704	776	859	1888		
EQ	000007E0'	198	197	2262				
EQO	00000A70'	245	244	1782	2114	2158	2168	
EQO2	00000AC0'	250	248					
EQ2	00000840'	204	202					
EXECUTE	00000FE0'	341	340	2242	2271	A1813	A1959	

```
LABEL                          VALUE       DEFN   REF

FAL                            00001470'    415    414   951  1336  2047
FALSE                          00000000      34    845 A1493 A1508 A1593
FBASE                          00006B80'   1548   1561
FBOUNDARY                      0000FB50'  A 937  A 969 A1114
FCOMPILER                      00002A50'    745
FCONTEXT                       00002610'    685
FCORE                          00002B10'    755    685   695
FCURBRUSH                      0000F070'  A 782  A 803
FCURRENT                       000026F0'    695
FCURSTENCIL                    0000EF90'  A 770  A 817 A 916 A 949 A1093
FDP                            00001810'    474    461   485   487   498   500
                                                   511   518   530   536   799
                                                   868   881  1100  1102  1104
                                                  1114  1558  1925 A 453 A 466
                                                 A1274 A1279 A1675 A1679 A1707
                                                 A1717 A1719 A1723 A1733
FETCH                          00000C10'    275    274   706   707   779   781
                                                   784   865   931  1057  1617
                                                  1689  1721  1765  1767  2118
                                                  2122  2123  2127  2129  2154
                                                  2155  2164  2165  2170  2177
                                                  2294  2300 A1530 A1610 A1630
                                                 A1634 A1658 A1808 A1812 A1815
                                                 A1833 A1951 A1955 A1966
FGFUNCTION                     0000BB20'  A  81  A 146
FGORIGIN                       0000DC30'  A 482  A 500 A 506 A 508
FGRAPHICS                      0000BA30'  A  68
FGRAPHMODE                     000122F0'  A1471  A1482 A1494 A1507 A1592
FILL                           000105B0'  A1066  A1065 A1080 A1972
FILL1                          00010740'  A1081  A1068 A1071
FILLMODE                       000124A0'  A1493  A1492
FIND                           000041E0'   1000    999  2156  2166  2179
FINP                           00007630'   1675    803   829  1929
FLAST                          00002BD0'    766
FMARK                          000057F0'   1335   1334  1385  1398  1453  1479
FMODE                          00003AC0'    920  A1701 A1858 A1897
FNENDBUF                       00009330'   1970    846  1934
FPAD                           000073F0'   1644   1931
FRESOLVE                       00005920'   1348   1347  1400  1410  1467  1497
                                                  1514
FROTATION                      00010FC0'  A1191  A1213
FSCALEXA                       00011AD0'  A1357  A1412
FSCALEXB                       00011AF0'  A1358  A1413
FSCALEYA                       00011CB0'  A1385  A1418
FSCALEYB                       00011CD0'  A1386  A1419
FSHEARXA                       00011BC0'  A1371  A1439
FSHEARXB                       00011BE0'  A1372  A1440
FSHEARYA                       00011DA0'  A1399  A1445
FSHEARYB                       00011DC0'  A1400  A1446
FSP0                           000095A0'   1996
FTIB                           00007570'   1665   1928
FTOBP                          000074B0'   1654    668   673  1932
FVORG                          0000BCC0'  A 107  A 123
FVTRANSP                       0000BC00'  A  94  A 121
F_ACHOU                        000043E0'   1028
F_ACHOU1                       00004410'   1031   1029
F_ACHOU2                       00004490'   1037   1035
F_COMPARA                      000042A0'   1010   1042
F_COMPCHAR                     00004370'   1021   1027
F_NAOACHOU                     000044F0'   1043   1006  1009  1040
F_PROXIMO                      000044B0'   1039   1016  1024
F_SAI                          00004530'   1046   1038
GADDSAT                        0000D3F0'  A 370  A 369
GAND                           0000C300'  A 178  A 177
GANDNOT                        0000C420'  A 190  A 189
GETCPT                         0000EC50'  A 728  A 727 A1531 A1631 A1834
GFUNCTION                      0000BAF0'  A  80  A  79
GIN                            00007600'   1674   1673
GMAX                           0000D720'  A 406  A 405
GMIN                           0000D820'  A 418  A 417
GMINUS                         0000D500'  A 382  A 381
GNAND                          0000D0B0'  A 334  A 333
GNOR                           0000C960'  A 250  A 249
```

LABEL	VALUE	DEFN	REF					
GNOTAND	0000CD80'	A 298	A 297					
GNOTD	0000C860'	A 238	A 237					
GNOTOR	0000CFA0'	A 322	A 321					
GNOTS	0000D1C0'	A 346	A 345					
GOR	0000CA60'	A 262	A 261					
GORIGIN	0000DC00'	A 481	A 480					
GORNOT	0000C640'	A 214	A 213					
GOUT	00007480'	1653	1652					
GPLUS	0000D2D0'	A 358	A 357					
GRAPHICS	0000BA00'	A 67	A 66					
GRAPHMODE	000122C0'	A1470	A1469	A1965				
GSUBSAT	0000D620'	A 394	A 393					
GT	00000BC0'	214	213					
GTO	00000B40'	260	259					
GTO2	00000B90'	265	263					
GT2	00000920'	220	218					
GTSHARP	00007F60'	1763	1762	1801				
GXNOR	0000C750'	A 226	A 225					
GXOR	0000CC60'	A 286	A 285					
HCOUNT	C00001B0	A 30						
HEBLNK	C0000010	A 31						
HERE	00001720'	461	460	860	969	1335	1348	
			1361	1701	2131	2152		
HESYNC	C0000000	A 32						
HLD	00007300'	1632	1631	1686	1688	1708	1764	
HOLD	000076C0'	1684	1683	1736	1752			
HSBLNK	C0000020	A 33						
HSTADRH	C00000E0	A 19						
HSTADRL	C00000D0	A 20						
HSTCTLH	C0000100	A 21						
HSTCTLL	C00000F0	A 22						
HTOTAL	C0000030	A 34						
H_ABORTQUOTE	000097D0'	2028	2042					
H_ABS	00005440'	1275	1287					
H_ADDRSHOW	00013CB0'	A1752	A1764					
H_ADDXY	00012A60'	A1560	A1578					
H_ADJUSTXY	00013D90'	A1764	A1776					
H_AGAIN	00006050'	1434	1446					
H_ALINETO	00012500'	A1502	A1518					
H_ALLOT	00001830'	478	492					
H_AMOVETO	00012E80'	A1604	A1618					
H_AND	00000470'	138	151					
H_ARRAY	00013590'	A1667	A1696					
H_ASCII	00006FD0'	1606	1833					
H_BASE	00006AE0'	1542	1553					
H_BEGIN	00005E70'	1415	1425					
H_BFETCH	00013E90'	A1776	A1791					
H_BLACK	0000C4B0'	A 197	A 209					
H_BMARK	000059C0'	1356	1367					
H_BOUNDARY	0000FA90'	A 931	A 944					
H_BRACKCOMPILE	0000AAF0'	2197	2208					
H_BRACKTICK	0000AC00'	2208	745					
H_BRANCH	00001480'	423	437					
H_BRESOLVE	00005A80'	1367	1378					
H_BSHOW	00013F60'	A1791	A1827					
H_BUILDS	00003E50'	960	979					
H_CBRANCH	00001570'	437	456					
H_CCOMMA	00001A40'	506	524					
H_CFETCH	00000CF0'	294						
H_CLEAVE	000069F0'	1530	1606					
H_CMISSING	00009F70'	2109	2147					
H_COLON	00003AE0'	925	995					
H_COMMA	00001950'	492	506					
H_COMMAQUOTE	00004830'	1092	1120					
H_COMPILATOR	0000B010'	2252	2285					
H_COMPILE	00001B70'	524	543					
H_COMPILER	00002990'	739	749					
H_COMPILERWORD	00002BF0'	771	794					
H_COMPOSE	0000DE80'	A 517	A 546					
H_CONSTANT	00003760'	887	900					
H_CONTEXT	00002550'	679	689					
H_CORE	00002A70'	749	760	771				
H_CREATE	00003380'	852	887					

```
00000278        52
0103 00000278                    .even
0104 00000280 000002A0'          .field over
```

LABEL	VALUE	DEFN	REF
H_CSTACK	00009B80'	2082	2109
H_CSTORE	00000DC0'	307	322
H_CURBMFONT	00013BC0'	A1740	A1752
H_CURBRUSH	0000EFB0'	A 776	A 798
H_CURCPT	00014490'	A1827	A1842
H_CURRENT	00002630'	689	700
H_CURSTENCIL	0000EEC0'	A 764	A 776
H_CURTFONT	000145B0'	A1842	A1853
H_DDUP	000117A0'	A1303	A1320
H_DEFINED	0000A490'	2147	2184
H_DETACH	0000E2E0'	A 591	A 633
H_DIFD	0000ACE0'	2219	2234
H_DIP	00001770'	468	478
H_DIV	00004F00'	1183	1199
H_DIVMOD	000050D0'	1215	1237
H_DO	00006410'	1472	1485
H_DOCONSTANT	000010D0'	361	374
H_DOCREATE	00001010'	348	361
H_DOES	00004050'	979	1141
H_DOSTENCIL	0000D8B0'	A 442	A 476
H_DOT	00008540'	1822	1882
H_DOTQUOTE	00004C20'	1141	1155
H_DOUBLEFETCH	000116E0'	A1288	A1303
H_DOUBLESTORE	00011880'	A1320	A1337
H_DOUBLEVAR	00011500'	A1268	A1288
H_DRAWMODE	00012310'	A1476	A1488
H_DROP	000000E0'	77	87
H_DROPTRUE	00014940'	A1876	A1892
H_DUP	00000030'	65	77
H_ELSE	00005C50'	1391	1405
H_ENTRY	00002710'	700	712
H_EQ	00000780'	193	209
H_EQD	00000A10'	240	255
H_EXECUTE	00000F50'	336	348
H_FAL	000013F0'	410	423
H_FETCH	00000BB0'	270	282
H_FILL	00010540'	A1061	A1088
H_FILLMODE	00012410'	A1488	A1502
H_FIND	00004160'	995	1051
H_FMARK	00005770'	1330	1343
H_FRESOLVE	00005B90'	1343	1356
H_GADDSAT	0000D360'	A 365	A 377
H_GAND	0000C290'	A 173	A 185
H_GANDNOT	0000C390'	A 185	A 197
H_GETCPT	0000EBD0'	A 723	A 744
H_GFUNCTION	0000BA50'	A 75	A 88
H_GIN	00007590'	1669	1679
H_GMAX	0000D6B0'	A 401	A 413
H_GMIN	0000D7B0'	A 413	A 442
H_GMINUS	0000D480'	A 377	A 389
H_GNAND	0000D030'	A 329	A 341
H_GNOR	0000C8F0'	A 245	A 257
H_GNOTAND	0000CCF0'	A 293	A 305
H_GNOTD	0000C7E0'	A 233	A 245
H_GNOTOR	0000CF20'	A 317	A 329
H_GNOTS	0000D140'	A 341	A 353
H_GOR	0000C9F0'	A 257	A 269
H_GORIGIN	0000DB70'	A 476	A 490
H_GORNOT	0000C5C0'	A 209	A 221
H_GOUT	00007410'	1648	1659
H_GPLUS	0000D250'	A 353	A 365
H_GRAPHICS	0000B970'	A 62	755 A 75
H_GRAPHMODE	00012220'	A1465	A1476
H_GSUBSAT	0000D590'	A 389	A 401
H_GT	00000860'	209	225
H_GTO	00000AE0'	255	270
H_GTSHARP	00007F00'	1758	1774
H_GXNOR	0000C6D0'	A 221	A 233
H_GXOR	0000CBF0'	A 281	A 293
H_HERE	000016B0'	456	468
H_HLD	00007290'	1627	1638 1696
H_HOLD	00007650'	1679	
H_I	00002020'	598	610

LABEL	VALUE	DEFN	REF
H_IF	00005B50'	1378	1391
H_IMMEDIATE	00004540'	1051	1069
H_INNER	0000B4F0'	2285	A 62
H_INPUT	00008D70'	1920	1964
H_INTERPRETER	0000ADC0'	2234	2252
H_J	000020B0'	610	621
H_K	00002150'	621	632
H_LAST	00002B30'	760	
H_LEAVE	00006900'	1519	1530
H_LEFTBRACKET	000098F0'	2042	2197
H_LEFTPAR	00013790'	A1696	A1740
H_LINEB	0000F090'	A 798	A 931
H_LINEF	00010750'	A1088	A1185
H_LINETO	00012B70'	A1578	A1604
H_LIT	000011B0'	374	388
H_LITERAL	00008610'	1833	1845
H_LOADFUNCTION	0000BF60'	A 140	A 161
H_LOOP	00006510'	1485	1502
H_LT	000006A0'	177	193
H_LTO	00000940'	225	240
H_LTSHARP	00007810'	1696	1715
H_MAX	000055B0'	1299	1314
H_MIN	00005690'	1314	1542
H_MINUS	000028E0'	725	739
H_MOD	00004FE0'	1199	1215
H_MODE	00003A20'	914	925
H_MOVETO	000132C0'	A1643	A1667
H_NEGATE	000054F0'	1287	1299
H_NENDBUF	00009270'	1964	1974
H_NOP	0000CAF0'	A 269	A 281
H_NOT	000003C0'	126	138
H_NUMBER	00006BA0'	1553	1627
H_OR	00000530'	151	164
H_ORIGIN	00012970'	A1547	A1560
H_OVER	00000230'	100	111
H_PAD	00007350'	1638	1648
H_PASTE	0000E020'	A 546	A 591
H_PLUS	00002830'	712	725
H_PLUSLOOP	00006700'	1502	1519
H_PLUSTORE	00000E90'	322	336
H_QUOTE	00004D20'	1155	1330
H_RABORT	00009350'	1974	1990
H_RABORTQUOTE	000095C0'	2000	2054
H_RCLEAVE	000022D0'	644	657
H_RDO	00001CE0'	543	565
H_RDOT	00008210'	1790	1822
H_RDOTQUOTE	00004A80'	1120	1167
H_REPASTE	0000E560'	A 633	A 676
H_REPEAT	00006290'	1458	1472
H_REPLACE	0000C170'	A 161	A 173
H_RETSTK	000099F0'	2054	2066
H_RFETCH	00008890'	1870	2028
H_RFROM	00008740'	1845	1858
H_RLEAVE	000021F0'	632	644
H_RLINETO	00012670'	A1518	A1547
H_RLOOP	00001DF0'	565	587
H_RMOVETO	00012FB0'	A1618	A1643
H_ROT	000002E0'	111	126
H_ROTATION	00010FE0'	A1193	A1208
H_ROTAVAR	00010F00'	A1185	A1193
H_RPLUSLOOP	00001F70'	587	598
H_RQUOTE	000046C0'	1069	1092
H_RROTATION	000110E0'	A1208	A1268
H_RTO	000087F0'	1858	1870
H_SCALE	00011DE0'	A1405	A1430
H_SCALEX	00011A20'	A1351	A1365
H_SCALEY	00011C00'	A1379	A1393
H_SELECTBMFONT	00014690'	A1853	A1876
H_SELECTTFONT	00014A70'	A1892	A1931
H_SEMI	00003CD0'	943	960
H_SERIAL	00008BC0'	1901	1920
H_SETCPT	0000ED50'	A 744	A 764
H_SFILL	0000FB70'	A 944	A1061

LABEL	VALUE	DEFN	REF				
H_SHARP	00007A20'	1715	1742				
H_SHARPS	000080C0'	1774	1790				
H_SHEAR	00011FE0'	A1430	A1465				
H_SHEARX	00011B10'	A1365	A1379				
H_SHEARY	00011CF0'	A1393	A1405				
H_SIGN	00007D80'	1742	1758				
H_SPC	00001290'	388	399				
H_SPO	00009500'	1990	2000				
H_STENCIL	0000DC50'	A 490	A 517				
H_STORE	00000C50'	282	294	307			
H_SWAP	00000170'	87	100				
H_THEN	00005DC0'	1405	1415				
H_TIB	000074D0'	1659	1669				
H_TICK	0000A9C0'	2184	2219				
H_TIMES	00004E20'	1167	1183				
H_TIMESDIV	00005210'	1237	1256				
H_TIMESDIVMOD	00005310'	1256	1275				
H_TOKEN	00002F20'	794	852				
H_TRANSFER	0000EBB0'	A 676	A 723				
H_TRANSPARENCY	0000BCE0'	A 115	A 140				
H_TRU	00001340'	399	410				
H_TSHOW	00014D20'	A1931	766	A 68			
H_TYPE	000023A0'	657	679				
H_ULT	00009AA0'	2066	2082				
H_UNTIL	00005F30'	1425	1434				
H_VARIABLE	000038C0'	900	914				
H_VCPT	00011940'	A1337	A1351				
H_VOCABULARY	00008930'	1882	1901				
H_VORG	0000BC20'	A 101	A 115				
H_VTRANSP	0000BB40'	A 88	A 101				
H_WHILE	00006170'	1446	1458				
H_WHITE	0000CE10'	A 305	A 317				
H_XOR	000005E0'	164	177				
I	00002080'	603	602	A1803	A1946		
IF	00005BB0'	1383	1382				
IMMEDIATE	000045E0'	1056	1055				
INC1	0000001B	A 52	A1132	A1143	A1145	A1153	A1154
			A1156				
INC2	0000001C	A 53	A1133	A1137	A1139	A1154	A1155
			A1156				
INNER	0000B570'	2290	1986	2289	2312		
INNER1	0000B600'	2293	2310				
INNER2	0000B840'	2306	2302				
INNER3	0000B870'	2307	2305				
INNER4	0000B920'	2311	2296				
INPUT	00008DF0'	1925	1924	2290			
INPUTLE	00009080'	1943	1940				
INPUTSAI	000091A0'	1953	1942	1946			
INPUTVERIF	00009010'	1938	1941	1948	1952		
INTENB	C0000110	21					
INTERPRET2	0000AF70'	2245	2241				
INTERPRET3	0000B000'	2248	2244				
INTERPRETER	0000AE70'	2239	2238	2306			
INTPEND	C0000120	A 24					
ISTK	0000000D	31	1983				
I_ASTK	00005200	37					
I_ISTK	00005200	38					
I_PSTK	00005200	36					
I_RSTK	00005200	39					
J	00002110'	615	614				
K	000021B0'	626	625				
LAST	00002BA0'	765	764	862	1056		
LEAVE	00006980'	1524	1523				
LEFTBRACKET	00009950'	2047	2046				
LEFTPAR	000137F0'	A1701	A1700				
LEFTPAR1	00013970'	A1715	A1703				
LEFTPAR2	00013B60'	A1731	A1729				
LEFTPAR3	00013BB0'	A1734	A1714				
LINEB	0000F110'	A 803	A 802	A1510	A1595		
LINEB1	0000F410'	A 837					
LINEB10	0000F860'	A 893					
LINEB11	0000F8C0'	A 900	A 911				
LINEB12	0000F950'	A 909	A 905				

LABEL	VALUE	DEFN	REF
LINEB13	0000F960'	A 910	A 908
LINEB2	0000F4C0'	A 846	
LINEB3	0000F620'	A 865	A 862
LINEB4	0000F650'	A 866	A 861 A 864
LINEB5	0000F6D0'	A 871	A 868
LINEB6	0000F700'	A 872	A 867 A 870
LINEB7	0000F7A0'	A 883	A 875
LINEB7A	0000F800'	A 887	A 885
LINEB8	0000F820'	A 889	
LINEB9	0000F840'	A 891	
LINEF	000107D0'	A1093	A1092 A1512 A1597
LINEF1	00010A10'	A1115	
LINEF2	00010AB0'	A1121	
LINEF3	00010C10'	A1139	A1136
LINEF4	00010C40'	A1140	A1135 A1138
LINEF5	00010CC0'	A1145	A1142
LINEF6	00010CF0'	A1146	A1141 A1144
LINEF7	00010D90'	A1156	A1149
LINETO	00012BF0'	A1583	A1582
LINETO1	00012E40'	A1597	A1594
LINETO2	00012E70'	A1598	A1596
LIT	00001230'	379	378 948 966 970 1493
			1510 1614 1621 1684 1702
			1724 1730 1733 1750 1839
			2096 2132 2265 2279 A 166
			A 178 A 190 A 202 A 214 A 226
			A 238 A 250 A 262 A 274 A 286
			A 298 A 310 A 322 A 334 A 346
			A 358 A 370 A 382 A 394 A 406
			A 418 A1799 A1804 A1939 A1942
			A1947
LITERAL	000086A0'	1838	1837 2214
LOADFUNCTION	0000C010'	A 145	A 144 A 168 A 180 A 192 A 204
			A 216 A 228 A 240 A 252 A 264
			A 276 A 288 A 300 A 312 A 324
			A 336 A 348 A 360 A 372 A 384
			A 396 A 408 A 420
LOOP	00006580'	1490	1489
LT	00000700'	182	181 1727
LTO	000009A0'	230	229 1747
LTO2	000009F0'	235	233
LT2	00000760'	188	186
LTSHARP	00007870'	1701	1700 1797
MAX	00005620'	1304	1303
MAX2	00005670'	1309	1307
MIN	00005700'	1319	1318
MIN2	00005750'	1324	1322
MINUS	00002940'	730	729 1769
MOD	00005050'	1204	1203
MODE	00003A90'	919	918 936 952 2048 2117
			2169 2299
MOVETO	00013340'	A1648	A1647
NEGATE	00005570'	1292	1291
NENDBUF	00009300'	1969	1968 2293
NOP	0000CB60'	A 274	A 273
NOT	00000430'	131	130
NUMBER	00006C20'	1558	1557 2246 2276
NUMBER1	00006E20'	1580	1576 1593
NUMBER2	00006EE0'	1588	1584
NUMBER3	00006F80'	1597	1595
NUMBERSAI1	00006FA0'	1599	1573 1582 1586 1589
OFFSET	00000014	A 44	A 656 A 693 A 705 A 819 A 951
			A1095
OR	00000590'	156	155
ORIGIN	000129F0'	A1552	A1551 A1589 A1654 A1957
OVER	000002A0'	105	104 1726 1768
PAD	000073C0'	1643	1642 1706 1766
PASTE	0000EDA0'	A 551	A 550 A1816
PATTRN	0000001D	A 50	A1103
PLUS	00002890'	717	716 1495 1512 1616 1704
			1732 1735 2098 2134 A1809
			A1952
PLUSLOOP	00006780'	1507	1506

LABEL	VALUE	DEFN	REF
PLUSTORE	00000EF0'	327	326 1687
PMASK	C0000160	A 17	
PSIZE	C0000150	A 16	
PSTK	0000000B	29	1324 1598 1599 1802 1817 1853 1864 1876 1981 2006 2008 2022 2059 2061 2072 2073 2077 2225 2229 A 120 A 145 A 448 A 449 A 450 A 451 A 452 A 455 A 469 A 495 A 496 A 497 A 498 A 499 A 501 A 502 A 503 A 505 A 522 A 530 A 551 A 566 A 596 A 609 A 638 A 639 A 644 A 645 A 650 A 681 A 682 A 687 A 699 A 728 A 736 A 737 A 749 A 750 A 751 A 837 A 838 A 845 A 915 A 972 A1022 A1047 A1066 A1069 A1077 A1078 A1115 A1116 A1121 A1169 A1219 A1220 A1223 A1224 A1255 A1256 A1258 A1259 A1282 A1293 A1296 A1297 A1308 A1309 A1310 A1311 A1312 A1313 A1325 A1326 A1327 A1410 A1411 A1423 A1424 A1435 A1436 A1452 A1453 A1532 A1533 A1534 A1535 A1538 A1539 A1565 A1566 A1567 A1568 A1571 A1572 A1673 A1674 A1680 A1683 A1686 A1705 A1712 A1713 A1721 A1781 A1784 A1882 A1885
QUOTE	00004D80'	1160	1159
RABORT	000093E0'	1979	44 1978 2021
RABORTQUOTE	00009650'	2006	2005 2035 2091 2100 2137
RABORTQUOTE1	00009710'	2015	2013
RABORTQUOTE2	000097B0'	2022	2019
RCLEAVE	00002360'	649	648 1536
RCR	02003000	25	
RDO	00001D50'	548	547 1478 A1801 A1944
RDOSTENCIL	00000B40'	A 468	
RDOT	00008280'	1795	1794 1827
RDOT1	00008470'	1811	1806
RDOTQUOTE	00004AF0'	1126	1125 1148
RDOTQUOTE1	00004BB0'	1136	1134
RDOUBLEVAR	000116B0'	A1281	A1342 A1356 A1370 A1384 A1398 A1552 A1769
REFCNT	C00001F0	A 23	
REPASTE	0000E5F0'	A 638	A 637
REPEAT	00006310'	1463	1462
REPLACE	0000C200'	A 166	A 165
RESET	00000000'	44	
RETSTK	00009A60'	2059	2058 2087 2094
RFETCH	000088F0'	1875	1874
RFROM	000087A0'	1850	1849 1891
RLEAVE	00002280'	637	636 651 1525
RLINETO	00012700'	A1523	A1522
RLOOP	00001E70'	570	569 1491 A1820 A1963
RLOOP1	00001E90'	571	593
RLOOP2	00001F20'	579	575
RMOVETO	00013040'	A1623	A1622 A1819 A1962
RMR	02002000	24	
ROT	00000350'	116	115 1723 1799
ROTATION	00011070'	A1198	A1197
ROTAVAR	00010F90'	A1190	A1189 A1198
RPLUSLOOP	00002000'	592	591 1508
RQUOTE	00004730'	1075	1074 1161 A1716
RQUOTE1	000047F0'	1085	1083
RRHR	02000000	22	1943
RROTATION	00011180'	A1213	A1212 A1528 A1588 A1628 A1653
RROTATION1	00011240'	A1222	A1216
RROTATION2	00011290'	A1225	A1221
RROTATION3	00011330'	A1233	A1245
RROTATION4	000114A0'	A1255	
RROTATION5	000114D0'	A1258	A1254
RROTATION6	000114F0'	A1260	A1215 A1257

LABEL	VALUE	DEFN	REF					
RSR	02001000	23	1938					
RSTK	0000000F	32						
RTO	00008850'	1863	1862					
SADDR	00000010	A 40	A 658	A 715	A 805	A 899	A 900	
			A 912	A1167				
SCALE	00011E60'	A1410	A1409	A1526	A1586	A1626	A1651	
SCALEX	00011AA0'	A1356	A1355					
SCALEY	00011C80'	A1384	A1383					
SELECTBMFONT	00014740'	A1858	A1857					
SELECTBMFONT1	00014840'	A1866	A1861					
SELECTBMFONT2	00014930'	A1871	A1865					
SELECTTFONT	00014B20'	A1897	A1896					
SELECTTFONT1	00014C20'	A1904	A1899					
SELECTTFONT2	00014D10'	A1909	A1903					
SEMI	00003D30'	948	947					
SERIAL	00008C40'	1906	1905	1985				
SETCPT	0000EDD0'	A 749	A 748	A1611	A1635	A1659		
SFILL	0000FBF0'	A 949	A 948	A1079				
SFILL1	0000FE60'	A 972	A1030	A1031				
SFILL10	000102C0'	A1025	A1005	A1006				
SFILL11	00010320'	A1029	A1037	A1049				
SFILL12	000103D0'	A1038	A1044					
SFILL13	00010460'	A1045	A1042					
SFILL14	000104D0'	A1050	A 974					
SFILL2	0000FE70'	A 973						
SFILL3	0000FEF0'	A 979	A 985					
SFILL4	0000FF80'	A 986	A 983					
SFILL5	0000FFF0'	A 991	A 997					
SFILL6	00010080'	A 998	A 995					
SFILL7	00010110'	A1004	A1012	A1024				
SFILL8	000101C0'	A1013	A1019					
SFILL9	00010250'	A1020	A1017					
SHARP	00007A80'	1720	1719	1779				
SHARP1	00007CC0'	1733	1729					
SHARPS	00008120'	1779	1778	1784	1798			
SHEAR	00012060'	A1435	A1434	A1527	A1587	A1627	A1652	
SHEARX	00011B90'	A1370	A1369					
SHEARY	00011D70'	A1398	A1397					
SIGN	00007DF0'	1747	1746	1800				
SIGN1	00007EF0'	1753	1749					
SPC	00001300'	393	392	857	1611	2189	2291	
			2307					
SPO	00009570'	1995	1994	2088	A1881			
SPTCH	00000011	A 41	A 659	A 714	A 811			
STENCIL	0000DCE0'	A 495	A 494					
STKSIZE	00005200	35	36	37	38	39	1980	
			1981	1982	1983	1996		
STORE	00000CB0'	287	286	782	786	788	863	
			866	933	937	953	987	
			1350	1707	1709	1894	1896	
			2049	2126	2130	A1199	A1798	
			A1864	A1870	A1902	A1908	A1938	
SWAP	000001E0'	92	91	785	986	1349	1399	
			1463	2089	A1796	A1936	A1941	
THEN	00005E30'	1410	1409					
TIB	00007540'	1664	1663					
TICK	0000AA20'	2189	2188	2202	2213	A1862	A1900	
TIMES	00004E80'	1172	1171	A 504	A1806	A1949		
TIMESDIV	00005270'	1242	1241					
TIMESDIVMOD	00005390'	1261	1260					
TOK1	00003070'	809	816					
TOK2	00003120'	817	814					
TOK3	00003140'	819	825					
TOK4	000031E0'	828	823					
TOK5	00003240'	831	834					
TOK6	000032C0'	840	836					
TOKEN	00002FA0'	799	798	858	1103	1612	2190	
			2292	2308	A1706	A1722		
TOKENBUF	00003320'	845	812					
TRANSFER	0000E940'	A 681	A 680					
TRANSPARENCY	0000BD90'	A 120	A 119					
TRU	000013B0'	404	403	935	1618	2136	2261	

LABEL	VALUE	DEFN	REF				
TRUE	FFFFFFFF	33	1033	1933	1970	A 937	A 973
			A1067	A1070	A1076	A1103	A1222
			A1253	A1481	A1883	A1940	
TSHOW	00014DA0'	A1936	A1935				
TSHOW1	00014F50'	A1946	A1964				
TSHOW2	000152C0'	A1965	A1945				
TSHOW3	000153F0'	A1972	A1968				
TSHOW4	00015420'	A1973	A1971				
TYPE	00002410'	662	661	1139	1828	2020	2135
TYPEMOVE	000024D0'	669	672				
TYPESAI	00002540'	674	667				
ULT	00009B00'	2071	2070	2090	2099		
ULT2	00009B60'	2077	2075				
UNTIL	0000 5FB0'	1430	1429				
VARIABLE	00003950'	905	904				
VCOUNT	C00001D0	A 35					
VCPT	000119B0'	A1342	A1341	A1524	A1584	A1624	A1649
VEBLNK	C0000050	A 36					
VESYNC	C0000040	A 37					
VOCABULARY	000089D0'	1887	1886				
VORG	0000BC90'	A 106	A 105				
VSBLNK	C0000060	A 38					
VTOTAL	C0000070	A 39					
VTRANSP	0000BBD0'	A 93	A 92				
WCR	02203000	28	1913	1937	1954		
WEND	00000016	A 46	A 821	A 953	A1097		
WHILE	000061F0'	1451	1450				
WHITE	0000CE90'	A 310	A 309				
WMR	02202000	27	1909	1911			
WSTART	00000015	A 45	A 836	A 968	A1101		
WTHR	02200000	26					
XOR	00000650'	169	168				

C:\TMS34010\TMS-ASM\STEFANI>

Ao Leitor

A autor está à disposição do leitor para sanar eventuais dúvidas, bem como para receber sugestões e críticas ao trabalho apresentado.

O autor poderá ser contactado nos seguintes endereços:

Sr. MARIO ANTONIO STEFANI

- Rua Machado Sidney, 11 - apto 1502
  São José dos Campos-SP-Brasil

  Fone: (0123) 22-4544

- OPTO ELETRONICA S/A
  Rua Joaquim A.R. de Souza, 1071
  São Carlos-SP-Brasil

  Fone: (0162) 72-3881