

Texto traduzido pelo Power Translator PRO

===== = = GUIA do GAVIN A
===== de ASSEMBLER 80x86 = = direitos
autorais (c) Estey de Gavin, 1995. Todos direitos reservaram.

Isto originalmente foi escrito para a Revista do Empresário de Games e depois de receber lotes de realimentação positiva que eu adicionei e expandiram. Gastei muito tempo trabalhar em ele e em eu apreciaria ouvir de você se gostasse de ele.

Se quer entrar em contato comigo então email me em:
gavin@senator.demon.co.uk ou em CompuServe 100767,1325

OS RECURSOS QUE SERIAM ----- ÚTIL--.

Há vários recursos que pode achar útil:

A lista de Instruções e coordenações:

Se tem TASM então o "Montador de Turbo de Borland Referência Rápida" tem uma lista de instruções e coordenações até 486. O "Pentium de Intel Operador Familiar Manual: Volume 3" são igualmente úteis.

A lista de Interromper:

Há vários livros que tenha esta informação mas mais para cima datar é Ralf Marrom Interrompe lista disponível livremente em quatro partes em ftp: //x2ftp oulu.fi/pub/msdos/programmin onde XX é a versão.

Um livro que cobre ambos estes temas e é uma referência útil de assembléia é "O guia de Revolutionary a Linguagem de montagem", ISBN 1-874416-12-5 publised por Prensa de WROX.

A VISTA GERAL DO 80x86 ----- FAMILIAR--.

A família 80x86 era primeira começada em 1981 com os 8086 e o novo membro é o Pentium que foi libertado treze anos mais tarde em 1994. São todos para trás compatíveis um com o outro mas cada novas gerações adicionou características e mais velocidade que a lasca prévia. Hoje há muito poucos computadores em uso que têm os 8088 e 8086 lascas neles como são muito outdated e lento. Há alguns 286's mas seus números estão inclinando

como hoje software torna-se mais e mais exigir. Regular os 386, primeiro CPU de 32 bits do Intel, agora está inclinando e parece que os 486 está agora a entrada sistema plano.

O Representation de números em ----- binário--.

Antes de nós comece a entender como programar na assembléia é melhor tentar entender como números são representados em computadores. Os números são armazenados em binário, baseia dois. Há vários termos que são usados para descrever números diferentes de tamanho e eu descreverei o que estes meio.

1 BIT: 0

Um bit é o pedaço simples de dados que existe. Seu é qualquer um um ou um zero.

1 NIBBLE: 0000 4 BITS

O nibble é quatro bits ou metade um byte. A nota que tem um valor máximo de 15 (1111 = 15). Isto é a base para o hexadecimal (baseia 16) sistema de número que é usado como é longe mais fácil entender. Os números de Hexadecimal vão de 1 a F e são seguido por um h declarar que o estão em hex. i.e. Fh = 15 decimal. Os números de Hexadecimal que começam com uma letra são prefixed com uns 0 (zero).

1 BYTE 00000000 2 NIBBLES 8 BITS

Um byte é 8 bits ou 2 nibbles. Um byte tem um valor máximo de FFh (255 decimal). Porque um byte é 2 nibbles o representation de hexadecimal é dois algarismos de hex em fileira i.e. 3Dh. O byte é também aquele tamanho dos registros de 8 bits que nós estaremos cobrindo mais tarde.

1 PALAVRA 0000000000000000 2 BYTES 4 NIBBLES 16 BITS

Uma palavra é dois bytes que são stuck junto. Uma palavra tem um valor máximo de (65,536 de FFFFh). Desde que uma palavra é quatro nibbles, é representado por quatro algarismos de hex. Isto é o tamanho dos registros de 16 bits.

----- De registros--.

Os registros são um lugar no CPU onde um número podem ser armazenados e

podem ser manipulados. Há três tamanhos de registros: de 8 bits, de 16 bits e em 386 e acima de 32 bits. Há quatro tipos de registros diferentes; registros gerais de propósito, registros de segmento, registros de índice e registros de pilha. Primeiramente eis descrições dos registros principais. Os registros de pilha e registros de segmento serão cobertos mais tarde.

Propósito geral Registra -----.

Estes são registros de 16 bits. Há quatro registros gerais de propósito; MACHADO, BX, CX e DX. São fendem para cima em registros de 8 bits. O MACHADO é fende para cima em AH que contem o byte alto e AL que contem o byte baixo. Em 386's e acima há registros também de 32 bits, estes têm os mesmos nomes como os registros de 16 bits mas com um 'E' em frente i.e. EAX. Pode usar AL, AH, MACHADO e EAX separadamente e os tratam como separa registros para algumas tarefas.

Se MACHADO conteve 24689 decimal:

AL DE AH 01100000 01110001

AH seria 96 e AL seriam 113. Se adicionasse um a AL seria 114 e AH seriam inalterados.

SI, DI, SP e BP também podem ser usados como registros gerais de propósito mas tem usos mais específicos. Eles não são fendem em dois halves.

O índice Registra -----.

Estes às vezes são chamados registros de pointer e eles são registros de 16 bits. Eles principalmente são usados para instruções de barbante. Há três SI de registros de índice (índice de fonte), DI (índice de destino) e IP (pointer de instrução). Em 386's e acima há registros também de 32 bits de índice: EDI e ESI. Você também pode usar BX a barbantes de índice. IP é um registro de índice mas nao podem ser manipulados diretamente como armazena o endereço da próxima instrução.

A pilha registra -----.

BP e SP são registros de pilha e são usados quando lida com a pilha. Serão cobertos quando conversamos sobre a pilha mais tarde.

Os segmentos e compensars -----.

Os desenhistas originais dos 8088 decidiram que ninguém jamais necessitará usar mais que um megabyte de memória então construíram a lasca então não podia aceder acima disso. O problema é aceder um megabyte inteiro 20 bits são necessitados. Os registros só têm 16 bits e eles não queriam usar dois porque isso seria 32 bits e eles pensaram que isto seria demais para anyone. Surgiram com o que eles pensaram era que um meio esperto resolvesse este problema: segmentos e compensars. Isto é um meio de fazer o endereçar com dois registros mas não 32 bits.

COMPENSAR = SEGMENTO * 16
 SEGMENTO = COMPENSAR / 16 (os 4 bits mais baixos são perdidos)

Um registro contém o segmento e outro registro contém o compensar. Se põe os dois registros junto você recebe um endereço de 20 bits.

O 0010010000010000 DE SEGMENTO -- - COMPENSAR ----0100100000100010 de 20 bits
 Endereça 00101000100100100010 ===== de DS = = ===== de SI = =

O aviso que DS e SI sobrepõem. Isto é como DS: SI é usado para fazer um 20 endereço de bit. O segmento está em DS e o compensar está em SI. A anotação normal para um Segmento/Compensar par é: SEGMENTO: COMPENSAR

Os registros de segmento são: CS, DS, ES, SS. Nos 386 + há também FS e GS.

Compensar registros são: BX, DI, SI, BP, SP, IP. Em 386 + modo protegido, QUALQUER registro geral (não um registro de segmento) pode ser usado como um Compensar registro. (Exceto IP, que você não pode manipular diretamente).

Se você agora estão pensando aquela assembléia está realmente duro e você não entende segmentos e compensars absolutamente então não preocupam-se. Eu não entendi-os a princípio mas lutei em e sabido que eles não eram tão duro de usar em prática.

O ----- DE PILHA--.

Como há só seis registros que são usados mais operações, você provavelmente estão perguntando-se como fazem-no evitar isso. É fácil. Há algo chamado uma pilha que é uma área de memória que você pode poupar e

poder restaurar valores a.

Isto é uma área de memória que é como uma pilha de pratos. O último põe em ser o primeiro que tomam fora. Isto às vezes é referido a como Dura Em Primeiro Fora (LOFO) ou Primeiro Em Primeiro para fora (LIFO). Se outro pedaço de dados é posto na pilha cresce para baixo.

Como pode ver a pilha começa num endereço alto e cresce para baixo. Assegura-se que você nao põe dados demais na pilha nem vai ir overflow.

UMA APRESENTAÇÃO A ----- DE INSTRUÇÕES DE ASSEMBLÉIA--.

Há muitas instruções na assembléia mas há só aproximadamente vinte que têm que saber e usarão muito freqüentemente. A maioria de instruções são compostas de três caracteres e tem um operand então uma vírgula então outro operand. Por exemplo por uns dados num registro você usa a instrução de MOV.

o machado de mov, 10; põem 10 em bx de mov de machado, 20; põem 20 em cx de mov de bx, 30; põem 30 em dx de mov de cx, 40; põem 40 em dx

O aviso que em alguma coisa de montador depois que um; (semicolon) é ignorado. Isto é muito útil para comentar seu código.

O EMPURRÃO E ESTOURO: DUAS INSTRUÇÕES USAR O ----- DE PILHA--.

Sabe sobre a pilha mas como nao por dados num para fora de ele. Há duas instruções simples que necessita saber: empurrão e estouro. Eis a sintaxe para seu uso:

O EMPURRÃO Põe um pedaço de dados sobre o topo da pilha

A sintaxe: dados de empurrão

O ESTOURO Põe o pedaço de dados do topo da pilha num registro especificado ou variável.

A sintaxe: registro de estouro ou variável

Este exemplo de código demonstra como usar o empurrão e instruções de estouro

o cx de empurrão; põe cx no machado de empurrão de pilha; põe machado no cx de estouro de pilha; põe valor de pilha em machado de estouro de cx; põe valor de pilha em machado

O aviso que os valores de CX e MACHADO serão trocados. Há uma instrução trocar dois registros: XCHG, que reduziria o fragmento prévio a "machado de xchg, cx".

TIPOS DE ----- DE OPERAND--.

Há três tipos de operands em montador: registro imediato e memória. Imediato é um número que será sabido em compilação e sempre será o mesmo por exemplo '20' ou "UM". Um operand de registro é qualquer propósito geral ou registro de índice por MACHADO de exemplo ou SI. Um operand de memória é um variável que é armazenado em memória que será coberto mais tarde.

ALGUMAS INSTRUÇÕES QUE VOCÊ NECESSITARÁ SABER

-----.

Isto é uma lista de algumas instruções importantes que você necessita saber antes de você pode entender ou pode escrever programas de assembléia.

MOV move um valor de um lugar a outro.

A sintaxe: destino de MOV, fonte

por exemplo: machado de mov, 10; move um valor imediato em bx de mov de machado, cx; move valor de cx em dx de mov de bx, Número; move o valor de Número em dx

INT chama um DOS ou função de BIOS que são sub-rotinas fazer coisas que nós bastante nao escreveria uma função para e.g. muda video modo, abre um arquivo etc.

A sintaxe: INT interrompe número

Por exemplo: 21h de int; Chamados DOS 10h de int de serviço; Chama o BIOS Video interromper

A maioria de interromper tem mais de uma função, este meio que você tem

que passar um número à função que você quer. Isto normalmente é posto em AH. Imprimir uma mensagem na tela todo que você necessita fazer é isto:

o ah de mov, 9; sub-rotina numera 9 21h de int; chama o interrompe

Mas primeiro tem que especificar o que imprimir. Esta função necessita DS: DX ser um pointer distante aonde a barbante é. A barbante tem que ser terminada com um sinal de dólar (\$). Isto seria fácil se DS podia ser manipulado diretamente, ficar redondo este temos que usar MACHADO.

Este exemplo mostra como funciona:

o dx de mov, COMPENSAR Mensagem; DX contem compensar de machado de mov de mensagem, Mensagem de SEG; MACHADO contem segmento de ds de mov de mensagem, machado; DS: pontas de DX a ah de mov de mensagem, 9; funcionam 9 - 21h de int de barbante de exposição; chamado dos serviço

As palavras COMPENSAM e SEG conta o compilador que você quer o segmento ou o compensar da mensagem põem no registro nao o conteúdo da mensagem. Agora sabemos para por para cima o código exibir a mensagem que nós necessitamos declarar a mensagem. No segment1 de dados nós declaramos a mensagem como isto:

DB de mensagem "Oi Mundo! \$"

O aviso que a barbante é terminado com um sinal de dólar. O que faz 'DB' meio? DB é curto para declarar byte e a mensagem é uma formação de bytes (caráter de ASCII toma para cima um byte). Os dados podem ser declarados num número de tamanhos: bytes (DB), palavras (DW) e palavras duplas (DD). Você nao tem que preocupar-se com palavras duplas no momento como necessita um registro de 32 bits, tal como EAX, assentá-los em.

Eis alguns exemplos de declarar dados:

Number1 db? Number2 dw?

A marca de pergunta (?) no meio de fim que os dados nao é iniciado i.e. não tem nenhum valor em começar com. Isso podia como facilmente é escrito como:

O db Number1 0 dw Number2 1

Desta vez Number1 é igual a 0 e Number2 é igual a 1 quando programa

cargas. Seu programa também será três bytes mais longo. Se declara um variável como uma palavra você não pode mover o valor deste variável num registro de 8 bits e você não pode declarar um variável como um byte e move o valor num registro de 16 bits. Por exemplo:

al,Number1 de mov; ax,Number1 de mov de ok; erro

bx,Number2 de mov; bl,Number2 de mov de ok; erro

Todo tem que lembrar-se de ser que você só pode por bytes em registros de 8 bits e palavras em registros de 16 bits.

SEU PRIMEIRO ----- DE PROGRAMA DE ASSEMBLÉIA--.

Agora que sabe algumas instruções básicas e um pouco sobre dados está tempo que nós olhamos num pleno programa de assembléia que pode estar compilado.

O alistamento 1: 1STPROG.ASM

; Isto é um programa simples que exhibe "Oi Mundo!" no; tela.

. modelo pequeno. pilha. dados

O db de mensagem "Oi Mundo! \$"; mensagem ser exposição

. dx de mov de código, COMPENSAR Mensagem; compensar de Mensagem está em machado de mov de DX, Mensagem de SEG; segmento de Mensagem está em ds de mov de MACHADO, machado; DS: pontas de DX a ah de mov de barbante, 9; funcionam 9 - 21h de int de barbante de exposição; chamado dos serviço

ax,4c00h de mov; retorno a dos DOS 21h de int

O FIM começa; fim aqui

----- DE INSTRUÇÕES DE COMPILAÇÃO--.

Estes são algumas instruções compilar e ligar programas. Se tem um compilador outro que TASM ou A86 então vê seu manual de instrução.

O Montador de Turbo:

[/t de arquivo de tlink de arquivo.asm de tasm]

O /interruptor de t faz um. arquivo de COM. Isto só trabalhará se o modelo de memória é declarado como minúsculo no arquivo de fonte.

A86:

o arquivo.asm a86

Isto compilará seu programa a um. arquivo de COM. Nao importa o que o modelo de memória é.

ALGUMAS INSTRUÇÕES QUE VOCÊ NECESSITA SABER

-----.

Isto é somente uma lista de algumas instruções básicas de assembléia que são muito importante e são usadas freqüentemente.

ADICIONE Adiciona o conteúdo de um número a outro

A sintaxe:

ADICIONE operand1,operand2

Isto adiciona operand2 a operand1. A resposta é armazenada em operand1. Dados imediatos nao podem ser usados como operand1 mas pode ser usado como operand2.

SUB Subtrae um número de outro

A sintaxe: operand1,operand2 de **SUB**

Isto subtrae operand2 de operand1. Dados imediatos nao podem ser usados como operand1 mas pode ser usado como operand2.

MUL Multiplica dois números de unsigned inteiros (sempre positivo) **IMUL** Multiplica dois números assinados inteiros (qualquer um negative positivo)

A sintaxe: registro de **MUL** ou registro variável de **IMUL** ou variável

Este **AL** de multiples ou **MACHADO** pelo registro ou variável dado. **AL** é multiplicado se um operand de sized de byte é dado e o resultado é armazenado em **MACHADO**. Se o operand é **MACHADO** de sized de palavra é

multiplicado e o resultado é colocado em DX: MACHADO.

Nuns 386, 486 ou Pentium o registro de EAX pode ser usado e a resposta é armazenada em EDX: EAX.

DIV Divide dois números de unsigned inteiros (sempre positivo) IDIV Divide dois números assinados inteiros (qualquer um negative positivo)

A sintaxe: registro de DIV ou registro variável de IDIV ou variável

Isto trabalha no mesmo meio como MUL e IMUL por dividir o número em MACHADO pelo registro ou variável dado. A resposta é armazenada em dois lugares. AL armazena a resposta e a sobra está em AH. Se o operand é um 16 registro de bit que o número em DX: MACHADO é dividido pelo operand e a resposta é armazenada em MACHADO e sobra em DX.

AS COISAS QUE FAZEM MAIS ----- FÁCIL--.

O meio nós entramos o endereço da mensagem que nós queremos imprimir era um cumbersome de bit. Tomou três linhas e não é a coisa fácil de lembrar-se de

o dx de mov, COMPENSAR machado de mov de MyMessage, ds de mov de MyMessage de SEG, machado

Podemos repor todo isto com somente uma linha. Isto faz o código mais fácil ler e ele mais fácil lembrar de.

o dx de mov, COMPENSAR MyMessage

Fazer isto trabalhar no começo de seu código adiciona estas linhas:

o machado de mov, @ds de mov de dados, machado

A nota: para A86 que você necessita mudar a primeira linha a:

o machado de mov, dados

Isto é porque todos os dados no segmento tem o mesmo valor de SEG. Por que isto em DS poupa-nos recarregar este cada tempo que nós queremos usar outra coisa no mesmo segmento.

----- DE ENTRADA DE TECLADO--.

Estamos indo usar interrompemos 16h, 00h de função ler o teclado. Isto recebe uma tecla da memória de teclado intermediária. Se não há um, espera até que há. Retorna o ESQUADRINHA código em AH e a tradução de ASCII em AL.

o ah de xor, ah; 00h de função - recebe 16h de int de caráter; interrompe 16h

Todo que nós necessitamos preocupar-se com para agora está o valor de ascii que está em AL.

A nota: XOR executa um Boolean Exclusivo OU. Comumente é usado para apagar um registro ou variável.

A IMPRESSÃO UM ----- DE CARÁTER--.

O problema é que temos a tecla que foi apertada em ah. Como exibimo-lo? Nós nao podemos usar 9h de função porque para que necessitamos já ter definido a barbante que tem que acabar com um sinal de dólar. Isto é o que nós fazemos contrariamente:

; depois que chamar 00h de função de interromper 16h

o dl de mov, al; movem al (código de ascii) em ah, 02h de mov de dl; 02h de função de interromper 21h de int 21h; chamado interrompe 21h

Se quer poupar o valor de AH então MACHADO de empurrão antes de e estoura-o depois.

CONTROLE ----- DE FLUÊNCIA--.

Na assembléia há um jogo de comandos para fluência de controle como em qualquer outra linguagem. Primeiramente o comando bem básico:

a etiqueta de jmp

Todo que isto faz mover à etiqueta especificado e começa executar o código aí. Por exemplo:

Jmp ALabel. . . ALabel:

O que fazemos se queremos comparar algo? Acabamos de receber uma tecla do operador mas queremos fazer algo com ele. Deixa algo de impressão para fora se é igual a algo mais. Como fazemos isso? É fácil. Usamos o pulo em comandos de condição. Eis uma lista deles:

PULE EM INSTRUÇÕES DE CONDIÇÃO:

Os pulos de JA se o primeiro número era JAE de número de acima do segundo mesmo como acima, mas também pulará se são pulos iguais de JB se os primeiros números estava embaixo o segundo JBE mesmo como acima, mas também pulará se são pulos iguais de JNA se os primeiros números NÃO estava acima (JBE) pulos de JNAE se o primeiro número NÃO estava acima nem o mesmo como (JNB) pulos de JNB [sbrk] se o primeiro número NÃO estava embaixo (JAE) pulos de JNBE se o primeiro número NÃO estava embaixo nem o mesmo como (JA) pulos de JZ se os dois números estavam JE igual mesmo como JZ, somente uns pulos diferentes de JNZ de nome se os dois números NÃO são JNE igual mesmo como acima pulo de JC se carrega bandeira é posto

A nota: o pulo só pode ser um máximo de 127 bytes em qualquer um direção.

A sintaxe: registro de CMP ou destino variável de jxx de valor

Um exemplo de isto é:

o al de cmp, 'Y'; compara o valor em al com ItsYES de je de Y; se é igual então pulo a ItsYES

Cada instrução toma para cima uma certa quantia de espaço de código. Receberá um aviso se tenta e pula mais de 127 bytes em qualquer um direção do compilador. Pode resolver isto por mudar uma sequência como isto:

O machado de cmp, 10; MACHADO é 10? o je feito; sim, deixa acabamento

a algo como isto:

O machado de cmp, 10; MACHADO é 10? o notdone de jne; não não é jmp feito; nós agora estamos feitos notdone:

Isto resolve o problema mas pode querer pensar sobre reorganizar seu código ou procedimentos que usam se isto acontece frequentemente.

Agora estamos indo olhar num programa que demonstra entrada, produção e fluência de controle.

O alistamento 2: PROGFLOW.ASM

; um programa demonstrar fluência de programa e entrada/produção. modelo minúsculo. 100h de org de código começa:

o dx de mov, COMPENSAR Mensagem; exibem uma mensagem no ah de mov de tela, 9; 21h de int 09h de função que usa; de interromper 21h

o dx de mov, COMPENSAR Incitam; exhibe uma mensagem no ah de mov de tela, 9; 21h de int 09h de função que usa; de interromper First_Time de jmp 21h

Prompt_Again: dx de mov, COMPENSA Outro; exhibe uma mensagem no ah de mov de tela, 9; 21h de int 09h de função que usa; de interromper 21h

First_Time: dx de mov, COMPENSAR Novamente; exibem uma mensagem no ah de mov de tela, 9; 21h de int 09h de função que usa; de interromper 21h

o ah de xor, ah; 00h de função de 16h de int; interrompe 16h recebe um bl de mov de caráter, al; poupa a bl

o dl de mov, al; movem al a ah, 02h de mov de dl; 02h de função - 21h de int de caráter de exposição; chamado DOS serviço

O bl de cmp, 'Y'; al é = Y? Prompt_Again de je; se sim então exhibe-o novamente bl de cmp, 'y'; al é = y? Prompt_Again de je; se sim então exhibe-o novamente

O TheEnd: dx de mov, COMPENSAR Adeus; impressão adeus ah de mov de mensagem, 9; usar funciona 9 21h de int; de interromper ah, 4Ch de mov 21h; termina 21h de int de DOS using de programa

. Equ de CR DE DADOS 13; entram equ de LF 10; linha-alimenta

DB de mensagem "UM Programa Simples de Entrada/Produção\$" CR Pronto de DB, LF," eis seu primeiro lembrete. \$" Novamente CR DE DB, LF," quer ser incitado novamente? \$" Outro CR DE DB, LF," eis outro lembrete! \$" Adeus CR DE DB, LF," Adeus então. \$" fim começa

A APRESENTAÇÃO A ----- DE
PROCEDIMENTOS--.

Na assembléia um procedimento é o equivalente a uma função em C ou para Pascal. Um procedimento proporciona um meio fácil a encapsulate algum cálculo que então pode ser usado sem preocupar-se como funciona. Com procedimentos que adequadamente são projetado-lo ignorar como um trabalho é feito.

Isto é como um procedimento é definido:

```
PROC AProcedure. . ; Algum código fazer algo. RET; se isto não está aqui  
então seu AProcedure DE ENDP de choque de vontade de computador
```

É igualmente fácil de correr um procedimento todo que você necessita fazer é isto:

chame AProcedure

Isto logo programa é um exemplo de como usar um procedimento. É como o primeiro exemplo nós olhamos em, todo que faz é impressão "Oi Mundo!" na tela.

O alistamento 3: SIMPPROC.ASM

```
; Isto é um programa simples demonstrar procedimentos. Devia; impressão  
Oi Mundo! no tela quando correu.
```

```
. modelo minúsculo. 100h de org de código
```

```
Comece: Display_Hi de chamado; Chama o ax,4C00h de mov de procedimento;  
retorno a DOS 21h de int; interrompe 4Ch de função 21h
```

```
O dx de mov de PROC Display_Hi, COMPENSAR OLÁ; põem compensar de mensagem  
em ah de mov de DX, 9; funcionam 9 - 21h de int de barbante de exposição;  
chamado DOS ENDP Display_Hi de ret de serviço
```

```
OLÁ DB "Oi Mundo! $"; define uma mensagem
```

o fim Começa

OS PROCEDIMENTOS QUE PASSAM ----- DE
PARÂMETROS--.

Os procedimentos não são tão úteis a menos que possa passar parâmetros modificar nem usar dentro do procedimento. Há três meios de fazer este e cobrirei todos três métodos: em registros, em memória e na pilha.

Há três programas de exemplo que todo realiza a mesma tarefa. Imprimem um bloco quadrado (ASCII estima 254) num lugar especificado. Os tamanhos dos arquivos quando compilado são: 38 para registro, 69 para memória e 52 para pilha.

Em ----- de registros--.

As vantagens de isto é que é fácil de fazer e de ser rápido. Todo que você tem que fazer é a ser move os parâmetros em registros antes de chamar o procedimento.

O alistamento 4: PROC1.ASM

; Este um procedimento imprimir um bloco na tela que usa; registros passar parâmetros (posição de cursor de onde a; imprime ele e cor).

. modelo minúsculo. 100h de org de código Começa: dh de mov, 4; fila imprimir caráter em dl de mov, 5; coluna imprimir caráter em al de mov, 254; valor de ascii de bloco exibir bl de mov, 4; cor exibir caráter

chame PrintChar; imprime nosso caráter

ax,4C00h de mov; termina 21h de int de programa

PROC de PrintChar PRÓXIMO cx de empurrão; poupa registros estar destruído

o bh de xor, bh; aclaram bh - página videa 0 ah de mov, 2; funcionam 2 - move 10h de int de cursor; fila e col são já em dx

o bx de estouro; restaura bh de xor de bx, bh; página de exposição - 0 ah de mov, 9; 09h de função escreve char & cx de mov de attrib, 1; exibem-no 10h uma de int de vez; serviço de bios de chamado

o cx de estouro; restaura ret de registros; retorno aonde foi chamado ENDP de PrintChar

o fim Começa

A PASSAGEM POR ----- DE MEMÓRIA--.

As vantagens deste método é que é fácil de fazer mas faz seu programa maior e pode ser mais lento.

Passar parâmetros por memória todo que você necessita fazer é cópia os a um variável que é armazenado em memória. Pode usar um variável no mesmo meio que você pode usar um registro mas comandos com registros são um lote mais rápido.

O alistamento 5: PROC2.ASM

; Este um procedimento imprimir um bloco na tela usar memória; passar parâmetros (posição de cursor de onde imprimir e; cor).

. modelo minúsculo. 100h de org de código Começa: Fila de mov, 4; fila imprimir Col de mov de caráter, 5; coluna imprimir caráter em Char de mov, 254; valor de ascii de bloco exibir Cor de mov, 4; cor exibir caráter

chame PrintChar; imprime nosso caráter

ax,4C00h de mov; termina 21h de int de programa

PROC de PrintChar PRÓXIMO bx de cx de machado de empurrão; poupa registros estar destruído

o bh de xor, bh; aclaram bh - página videa 0 ah de mov, 2; funcionam 2 - move dh de mov de cursor, dl de mov de Fila, 10h de int de Col; serviço de Bios de chamado

o al de mov, bl de mov de Char, bh de xor de Cor, bh; página de exposição - 0 ah de mov, 9; 09h de função escreve char & cx de mov de attrib, 1; exibem-no 10h uma de int de vez; serviço de bios de chamado

o machado de cx de bx de estouro; restaura ret de registros; retorno aonde foi chamado ENDP de PrintChar

Reme db? ; Variables armazenar db de Col de dados? O db de cor? Char db?

o fim Começa

A passagem por ----- de Pilha--.

Isto é o método flexível bem poderoso de parâmetros de passagem o

problema é que mais é complicado.

O alistamento 6: PROC3.ASM

; Este um procedimento imprimir um bloco na tela que usa o; pilha passar parâmetros (posição de cursor de onde imprimir; e cor).

. modelo minúsculo. 100h de org de código Começa: dh de mov, 4; fila imprimir barbante em dl de mov, 5; coluna imprimir barbante em al de mov, 254; valor de ascii de bloco exibir bl de mov, 4; cor exibir caráter

o bx de machado de dx de empurrão; põe parâmetros sobre o PrintString de chamado de pilha; imprime nosso dx de machado de bx de estouro de barbante; restaura registros

ax,4C00h de mov; termina 21h de int de programa

PROC de PrintString PRÓXIMO bp de empurrão; poupa bp de mov de bp, sp; põe sp em cx de empurrão de bp; poupa registros estar destruído

o bh de xor, bh; aclaram bh - página videa 0 ah de mov, 2; funcionam 2 - move dx de mov de cursor, [bp+8]; restaura 10h de int de dx; serviço de bios de chamado

o machado de mov, [bp+6]; bx de mov de caráter, [bp+4]; bh de xor de atributo, bh; página de exposição - 0 ah de mov, 9; 09h de função escreve char & cx de mov de attrib, 1; exibem-no 10h uma de int de vez; serviço de bios de chamado

o cx de estouro; restaura ret de bp de estouro de registros; retorno aonde foi chamado ENDP de PrintString

o fim Começa

Receber um parâmetro da pilha todo que você necessita fazer é elabora onde é. O último parâmetro está em BP + 2 e então o próximo e BP + 4.

O QUE SÃO MODELOS DE MEMÓRIA? -----.

Temos usado o. directive MODELO especificar o que tipo de modelo de memória que nós usamos, mas o que isto querem dizer?

A sintaxe:. MemoryModel MODELO

Onde MemoryModel pode ser PEQUENO, COMPACTO, MÉDIO, GRANDE, ENORME, MINÚSCULO OU PLANO.

Minúsculo

Este meio que há só um segmento para tanto código como dados. Este tipo de programa pode ser um. arquivo de COM.

Pequeno

Este meio que por omissão todo código é lugar em um segmento e todos dados declarado no segmento de dados também é colocado em um segmento. Este meio que todos procedimentos e variables estão endereçados como PRÓXIMO por apontar em compensars só.

Compacto

Este meio que por omissão todos elementos de código são colocados em um segmento mas cada elemento de dados pode ser colocado no próprio segmento físico. Este meio que elementos de dados são endereçados por apontar em ambos no segmento e compensar endereços. Os elementos de código (procedimentos) estão PRÓXIMO e variables estão DISTANTE.

Médio

Isto é o oposto a compacto. Os elementos de dados estão PRÓXIMO e procedimentos estão DISTANTES.

Grande

Este meio que tanto procedimentos como variables são DISTANTE. Tem que apontar em ambos o segmento e compensar endereços.

Plano

Isto nao é usado muito como é para 32 espaço de memória de unsegmented de bit. Para este necessita um DOS extender. Isto é o que você teria que usar se estivessem escrevendo um programa a interface com um C/C ++ programa que usou um DOS extender tal como DOS4GW ou PharLap.

MACROS (em Montador de Turbo) -----.

(Todos exemplos de código dado são para macros em Montador de Turbo.)

O Macros são muito útil para fazer algo que é feito freqüentemente mas pelo qual um procedimento nao poder ser uso. O Macros são substituídos quando o programa é compilado ao código que eles contêm.

Isto é a sintaxe para definir um macro:

O nome_de_macro de macro;; uma sequência de instruções; endm

Estes dois exemplos são para macros que toma longe o trabalho enfadonho de empurrar e estourar certos registros:

O endm de dx de estouro de cx de estouro de bx de estouro de machado de estouro de macro de SaveRegs

O machado de estouro de bx de estouro de cx de estouro de dx de estouro de macro de RestoreRegs

endm

A nota que os registros são estourados na ordem inversa a eles foram empurrados. Usar um macro em você programá-lo acaba de usar o nome do macro como uma instrução costumeira:

O SaveRegs; algum outro RestoreRegs de instruções

Este exemplo mostra como pode usar um macro poupar bater em. Este macro simplesmente imprime para fora uma mensagem à tela.

SomeText de macro de OutMsg PrintMe local, SkipData de jmp de SkipData

PrintMe db SomeText, '\$'

O SkipData: dx de mov de ds de estouro de cs de ds de dx de machado de empurrão, COMPENSAR cs: ah de mov de PrintMe, 9 endm de machado de dx de ds de estouro 21h de int

endm

Os único problemas com macros é que se você overuse que os leva a ele programar aumenta e maior e que tem problemas com múltipla definição de etiquetas e variables. O meio correto resolver este problema é usar o

directive LOCAL para declarar nomes macros interno.

A sintaxe: nome de LOCAL

Onde nome é o nome de uma etiqueta variável local.

O Macros com ----- de parâmetros--.

Outra propriedade útil de macros é que podem ter parâmetros. O número de parâmetros só é restringido pelo comprimento da linha.

A sintaxe:

O nome_de_par1,par2,par3 de macro de Macro;; comandos vão aqui; endm

Isto é um exemplo que adiciona os primeiros e segundos parâmetros e põe o resulta no terceiro:

O machado de empurrão num1,num2,result de macro de AddMacro; poupa machado de destruir ax,num1 de mov; põe num1 em machado adicionar ax,num2; adiciona num2 a ele resultado de mov, machado; movem resposta em machado de estouro de resultado; restaura endm de machado

OS ARQUIVOS E COMO USÁ-LOS -----.

Os arquivos podem ser abertos, lêem e escrito a. DOS tem alguns meios de fazer que isto que nos poupa o problema de escrito o próprias rotinas. Sim, mais interrompe. Eis uma lista de funções úteis de interromper 21h aquele lida com arquivos.

A nota: Bits são numerados de direito deixar.

Funcione 3Dh: abre arquivo

Abre um arquivo existente para ler, escrito ou anexar na rodada especificada e filename.

A ENTRADA: AH = AL 3Dh = morde 0-2 modo de Acesso 000 = lêem só 001 = escrevem só 010 = bits de lê/escreve 4-6 modo que compartilha (DOS 3 +) 000 = modo de compatibility 001 = negam todos 010 = nega escreve 011 = nega lê 100 = não nega nenhum DS: DX = segmento: compensar de pathname de ASCIIZ

A PRODUÇÃO: CF = 0 função é MACHADO de succesful = CF de cabo = 1 erro
ocorreu MACHADO = 01h de código de erro arquivo compartilhar arquivo
ausente 02h de software caminho nao achado 03h arquivo nao achado nao
existe nenhum cabo 04h acesso disponível 05h modo negado de acesso 0Ch
nao permitido

O que ASCIIZ quer dizer? Uma barbante de ASCIIZ como uma barbante de
ASCII com um zero no fim em vez de um sinal de dólar.

Importante: Lembra-se de poupar o arquivo manipular é necessitado para
mais tarde.

Como poupar o cabo de arquivo

É importante poupar o cabo de arquivo porque isto é necessitado fazer
alguma coisa com o arquivo. Bem como isto é feito? Há dois métodos que
nós podíamos usar: copia o cabo de arquivo em outro registro e nao usa
que registra nem copia a um variável em memória.

As desvantagens com o primeiro método é que você não terá que lembrar-se
de usar o registro que você poupou ele em e desperdiça um registro que
pode estar usado para algo mais útil. Estamos indo usar o segundo. Isto é
como é feito:

DW de FileHandle 0; usam isto para poupar o cabo de arquivo. . .
FileHandle de mov, machado; poupam o cabo de arquivo

Funcione 3Eh: fecha arquivo

Fecha um arquivo que foi aberto.

A ENTRADA: MACHADO = BX 3Eh = cabo de arquivo

A PRODUÇÃO: CF = 0 função é MACHADO bem de sucedido = CF destruído = 1
função MACHADO nao bem de sucedido = código de erro - arquivo 06h cabo
nao aberto de unauthorised.

Importante: Nao chama esta função com um zero cabo porque isso fechará a
entrada normal (o teclado) e você nao será capaz de entrar alguma coisa.

3Fh de função: lê arquivo/artifício

Lê bytes de um arquivo ou artifício a uma memória intermediária.

A ENTRADA: AH = BX 3Fh = CX de cabo = número de bytes ser lido DS: DX = segmento: compensar de uma memória intermediária

A PRODUÇÃO: CF = 0 função é MACHADO bem de sucedido = número de bytes lê
CF = 1 um erro ocorreu acesso 05h 06h negado cabo ilegal ou arquivo não aberto

Se CF = 0 e MACHADO = 0 então o pointer de arquivo era já no fim do arquivo e mais não pode ser lido. Se CF = 0 e MACHADO é menores que CX então só parte foi lida porque o fim do arquivo foi alcançado ou um erro ocorrido.

Esta função também pode ser usada para receber entrada do teclado. Use um cabo de 0, e para leitura depois que o primeiro retorno de carruagem, ou vez uma um número especificado de caracteres foi lido. Isto é um bom e método fácil de usar só deixar o operador entrar uma certa quantia de caracteres.

O alistamento 7: READFILE.ASM

; um programa demonstrar criar um arquivo e então escrito a; ele

. modelo pequeno. pilha. código

o machado de mov, @dados; endereço de base de ds de mov de segmento de dados, machado; põem isto em ds

O dx de mov, COMPENSAR FileName; põem endereço de filename em al de mov de dx, 2; modo de acesso - lê e escreve ah,3Dh de mov; 3Dh de função - abre um 21h de int de arquivo; chamado DOS Cabo de mov de serviço, machado; poupam cabo de arquivo para mais ErrorOpening de jc de tarde; pulo se carrega jogo de bandeira - erro!

O dx de mov, compensar Memória intermediária; endereço de memória intermediária em bx de mov de dx, Cabo; cabo em cx de mov de bx, 100; quantia de bytes ser lida ah,3Fh de mov; 3Fh de função - lê de 21h de int de arquivo; chamado dos ErrorReading de jc de serviço; pulo se carrega jogo de bandeira - erro!

o bx de mov, Cabo; põem cabo de arquivo em ah,3Eh de mov de bx; 3Eh de função - fecha um 21h de int de arquivo; chamado DOS serviço

o cx de mov, 100; comprimento de si de mov de barbante, COMPENSAR Memória intermediária; DS: SI - endereço de bh de xor de barbante, bh; página videa - 0 ah,0Eh de mov; 0Eh de função - escreve caráter

O NextChar: lodsb; AL = próximo caráter em 10h de int de barbante; NextChar de volta de serviço de BIOS de chamado

ax,4C00h de mov; termina 21h de int de programa

O ErrorOpening: dx de mov, compensar OpenError; exibem um ah,09h de mov de erro; 21h de int 09h de função que usa; chamado DOS ax,4C01h de mov de serviço; programa de fim com um errorlevel =1 21h de int

O ErrorReading: dx de mov, compensar ReadError; exibem um ah,09h de mov de erro; 21h de int 09h de função que usa; chamado DOS serviço

ax,4C02h de mov; programa de fim com um errorlevel =2 21h de int

. dados

Manipule DW? ; armazenar "C:\test.txt",0 DE DB de FileName de cabo de arquivo; arquivo ser aberto

DB de OpenError "Um erro tem occurred(opening)!\$" DB de ReadError "Um erro tem occurred(reading)!\$"

A memória DB intermediária 100 dup (?) ; memória intermediária armazenar dados

O FIM

3Ch de função: Cria arquivo

Cria um novo arquivo vazio numa rodada especificada com um pathname especificado.

A ENTRADA: AH = CX 3Ch = atributo de arquivo morde 0 = 1 bit de arquivo de lê-só 1 = 1 bit escondido de arquivo 2 = 1 bit de arquivo de sistema 3 = 1 volume (ignorado) morde 4 = 1 reservado (0) - guia morde 5 = 1 bit de arquivo morde 6-15 reservado (0) DS: DX = segmento: compensar de pathname de ASCIIZ

A PRODUÇÃO: CF = 0 função é MACHADO bem de sucedido = CF de cabo = 1 um erro ocorreu caminho 03h 04h não achado nenhum acesso disponível 05h de cabo negado

Importante: Se um arquivo do mesmo nome existe então será perdido. Assegure-se que não há nenhum arquivo do mesmo nome. Isto pode ser feito com a função embaixo.

4Eh de função: acha primeiro arquivo que combina

Procura o primeiro arquivo que combina o filename dado.

A ENTRADA: AH = CX 4Eh = máscara de atributo de arquivo (bits podem ser combinados) bit 0 = 1 bit 1 = 1 bit escondido 2 = 1 bit de sistema 3 = 1 bit de etiqueta de volume 4 = 1 bit de guia 5 = 1 bit de arquivo 6-15 DS reservado: DX = segmento: compensar de pathname de ASCIIZ

A PRODUÇÃO: CF = 0 função é sucedido bem [DTA] Área de Transferência de Disco = bloco de dados de FindFirst

O bloco de DTA

Compense Tamanho em bytes Quer dizer

0 21 Reservaram 21 1 Arquivo atribui 22 2 Tempo de durar modificou 24 2 Data durar modificou 26 4 Tamanho de arquivo (em bytes) 30 13 nome de Arquivo (ASCIIZ)

Um exemplo de verificar se arquivo existe:

Archive "C:\file.txt",0 DE DB; nome de arquivo que nós queremos

o dx de mov, COMPENSAR Arquivo; endereço de cx,3Fh de mov de filename; 3Fh de máscara de arquivo - qualquer ah,4Eh de mov de arquivo; 4Eh de função - acha primeiro 21h de int de arquivo; chamado DOS NoFile de jc de serviço

; arquivo de ditado de mensagem de impressão existe NoFile;; continua com criar arquivo

Isto é um exemplo de criar um arquivo e então escrito a ele.

O alistamento 8: CRIA.ASM

; Este programa de exemplo cria um arquivo e então escreve a ele.

. modelo pequeno. pilha. código

o machado de mov, @dados; endereço de base de ds de mov de segmento de dados, machado; põem-no em ds

o dx de mov, compensar StartMessage; exibem o ah,09h de mov de mensagem que começa; 21h de int 09h de função que usa; chamado dos serviço

o dx de mov, compensar FileName; põem compensar de filename em cx de xor de dx, cx; aclara cx - faz ah,3Ch costumeiro de mov de arquivo; 3Ch de função - cria um 21h de int de arquivo; chamado DOS CreateError de jc de serviço; pulo se há um erro

o dx de mov, compensar FileName; põem compensar de filename em al de mov de dx, 2; modo de acesso -lê e escreve ah,3Dh de mov; 3Dh de função - abre o 21h de int de arquivo; chamado dos OpenError de jc de serviço; pulo se há um Cabo de mov de erro, machado; poupa valor de cabo

o dx de mov, compensar WriteMe; endereço de informação escrever bx de mov, Cabo; cabo de arquivo para cx de mov de arquivo, 38 ;38 bytes ser escritos ah,40h de mov; 40h de função - escreve arquivar 21h de int; chamado dos WriteError de jc de serviço; pulo se há um machado de cmp de erro, cx; era todos os dados [sbrk] Escrito? WriteError de jne; não não era - erro!

o bx de mov, Cabo; põem cabo de arquivo em ah,3Eh de mov de bx; 3Eh de função - fecha um 21h de int de arquivo; chamado dos serviço

o dx de mov, compensar EndMessage; exibem o ah,09h final de mov de mensagem; 21h de int 09h de função que usa; chamado dos serviço

O ReturnToDOS: ax,4C00h de mov; termina 21h de int de programa

O WriteError: dx de mov, compensar WriteMessage; exibem um EndError de jmp de mensagem de erro

O OpenError: dx de mov, compensar OpenMessage; exibem um EndError de jmp de mensagem de erro

O CreateError: dx de mov, compensar CreateMessage; exibem uma mensagem de erro

O EndError: ah,09h de mov; 21h de int 09h de função que usa; chamado dos ax,4C01h de mov de serviço; termina 21h de int de programa

. dados

O equ de CR 13 equ de LF 10

DB de StartMessage "Este programa cria um arquivo NOVO.TXT chamado" DB," na rodada de C. \$" CR DE DB de EndMessage, LF," Arquivo cria OK, olha em arquivo a" DB," está seguro. \$"

DB de WriteMessage "Um erro ocorreu (WRITING) \$" DB de OpenMessage "Um erro ocorreu (OPENING) \$" DB de CreateMessage "Um erro ocorreu (CREATING) \$"

DB de WriteMe "OI, ISTO É UMA PROVA, FUNCIONOU? ",0 ?

"C:\new.txt",0 DE DB de FileName; nome de arquivo abrir DW de Cabo? ; armazenar cabo de arquivo

O FIM

Isto é um exemplo de como anular um arquivo depois que verificar que existe:

O alistamento 9: DELFILE.ASM

; Uma demonstração de como anular um arquivo. O novo.txt de arquivo em; c: é anulado (este arquivo é criado por cria.exe). Nós também; verificar se as saídas de arquivo antes de tentar anulá-lo

. modelo pequeno. pilha. dados

O equ de CR 13 equ de LF 10

Arquive "C:\new.txt",0 de db

Db anulado "c Anulado de arquivo: "c:\new.txt de db de NoFile \new.txt \$" não faz saídas - exiting \$" db de ErrDel "Nao pode anular arquivo - provavelmente escreve protegido \$"

. machado de mov de código, @dados; armam ds como o segmento para ds de mov de dados, machado; machado de uso como nós não podemos fazê-lo diretamente

o dx de mov, COMPENSAR Arquivo; endereço de filename procurar cx,3Fh de mov; 3Fh de máscara de arquivo - qualquer ah,4Eh de mov de arquivo; 4Eh de função - acha primeiro 21h de int de arquivo; chamado dos FileDontExist de jc de serviço

o dx de mov, COMPENSAR Arquivo; DS: pontas de DX arquivar ser matadas ah,41h de mov; 41h de função - anula 21h de int de arquivo; chamado DOS ErrorDeleting de jc de serviço; pulo se havia um erro

jmp EndOk

O EndOk: dx de mov, COMPENSAR Anularam; Endit de jmp de mensagem de exposição

O ErrorDeleting: dx de mov, COMPENSAR ErrDel; Endit de jmp de mensagem de exposição

O FileDontExist: dx de mov, COMPENSAR NoFile; mensagem de exposição

O EndIt: ah de mov, 9 21h de int

ax,4C00h de mov; termina programa e saída a DOS 21h de int; chamado DOS serviço

o fim

USAR O FINDFIRST E FINDNEXT FUNCIONA

-----.

O alistamento 10: DIRC.ASM

; Este programa demonstra como procurar arquivos. Imprime; para fora os nomes de todos os arquivos no c: \rodada e nomes de; os sub-guias

. modelo pequeno. pilha. dados

"C:*.*",0 de db de FileName; db de DTA de nome de arquivo 128 dup (?) ; memória intermediária armazenar o db de ErrorMessage de DTA "Um Erro ocorreu - exiting. \$"

. machado de mov de código, @dados; armam ds ser semelhante ao ds de mov, um; es de mov de segmento de dados, machado; também es

o dx de mov, COMPENSAR DTA; DS: DX aponta a ah,1AH de mov de DTA; 1Ah de função - 21h fixo de int de DTA; chamado DOS serviço

cx,3Fh de mov; máscara de atributo - todo arquiva dx de mov, COMPENSAR FileName; DS: ah,4Eh de mov de filename de ASCIZ de pontas de DX; 4Eh de função - acha primeiro 21h de int; chamado DOS erro de jc de serviço; pulo se carrega bandeira é posto

O LoopCycle: dx de mov, COMPENSAR FileName; DS: pontas de DX arquivar ah,4Fh de mov de nome; 4fh de função - acha próximo 21h de int; chamado DOS saída de jc de serviço; saída se carrega bandeira é posto

o cx de mov, 13; comprimento de si de mov de filename, COMPENSAR DTA + 30; DS: pontas de SI a filename em bh de xor de DTA, bh; página videa - 0 ah,0Eh de mov; 0Eh de função - escreve carácter

O NextChar: lodsb; AL = próximo carácter em 10h de int de barbante; NextChar de volta de serviço de BIOS de chamado

o di de mov, COMPENSAR DTA + 30; ES: pontas de DI a cx de mov de DTA, 13; comprimento de al de xor de filename, al; enchem com zero stosb de rep; apaga DTA

LoopCycle de jmp; continua procurar

o erro: dx de mov, COMPENSAR ErrorMessage; ah de mov de mensagem de erro de exposição, 9 saída 21h de int: ax,4C00h de mov; saída a DOS 21h de int

o fim

----- DE INSTRUÇÕES DE BARBANTE--.

Na assembléia há algumas instruções muito úteis para lidar com barbantes. Eis uma lista das instruções e a sintaxe para usar os:

MOV* Move Barbante: move byte, palavra ou palavra dupla em DS: SI a ES: DI

A sintaxe:

o movsb; move movsw de byte; move movsd de palavra; move palavra dupla

CMPS* barbante de Comparação: byte de comparações, palavra ou palavra dupla em DS: SI a ES: DI

A sintaxe:

o cmpsb; cmpsw de byte de comparação; cmpsd de palavra de comparação; compara palavra dupla

A nota: Esta instrução normalmente é usada com o prefix de REP.

SCAS* barbante de Busca: procura AL, MACHADO, ou EAX em barbante em ES: DI

A sintaxe:

o scasb; procura scasw de AL; procura scasd de MACHADO; procura EAX

A nota: Esta instrução normalmente é usada com o REPZ ou prefix de REPNZ.

Prefix de REP para instrução de barbante repete vezes de CX de instrução

A sintaxe:

rep StringInstruction

STOS* Move byte, palavra ou palavra dupla de AL, MACHADO ou EAX a ES: DI

A sintaxe:

o stosb; move AL em ES: stosw de DI; move MACHADO em ES: stosd de DI;
move EAX em ES: DI

DIRIGIU* Move byte, palavra ou palavra dupla de DS: SI a AL, PARA MACHADO ou EAX

A sintaxe:

o lodsb; move ES: DI em lodsw de AL; move ES: DI em lodsd de MACHADO;
move ES: DI em EAX

O alistamento 11: BARBANTES.ASM

; Este programa demonstra exemplos de barbante

. modelo pequeno. pilha. código

o machado de mov, @dados; pontas de machado a de ds de mov de segmento de dados, machado; põe-o em es de mov de ds, machado; põe-o em es também

o ah de mov, 9; funcionam 9 - dx de mov de barbante de exposição, COMPENSAR Message1; ds: pontas de dx a 21h de int de mensagem; chamado dos função

o cld; aclara bandeira de direção

o si de mov, COMPENSAR String1; fazem ds: ponta de si a di de mov String1, COMPENSAR String2; faz es: ponta de di a cx de mov String2, 18; comprimento de movsb de rep de barbantes; string1 de cópia em string2

o ah de mov, 9; funcionam 9 - dx de mov de barbante de exposição, COMPENSAR Message2; ds: pontas de dx a 21h de int de mensagem; chamado dos função

o dx de mov, COMPENSAR String1; 21h de int String1 de exposição; chamado DOS serviço

o dx de mov, COMPENSAR Message3; ds: dx aponta a 21h de int de mensagem; chamado dos função

o dx de mov, COMPENSAR String2; 21h de int String2 de exposição; chamado DOS serviço

o si de mov, COMPENSAR Diff1; fazem ds: ponta de si a di de mov Diff1, COMPENSAR Diff2; faz es: ponta de di a cx de mov Diff2, 39; comprimento de cmpsb de repz de barbantes; Not_Equal de jnz de barbantes de comparação; pulo se eles não são o mesmo

o ah de mov, 9; funcionam 9 - dx de mov de barbante de exposição, COMPENSAR Message4; ds: pontas de dx a 21h de int de mensagem; chamado dos função

jmp Next_Operation

Not_Equal: ah de mov, 9; funcionam 9 - dx de mov de barbante de

exposição, COMPENSAR Message5; ds: pontas de dx a 21h de int de mensagem;
chamado dos função

Next_Operation: di de mov, COMPENSAR SearchString; fazem es: ponta de di
a cx de mov de barbante, 36; comprimento de al de mov de barbante,' H';
caráter procurar scasb de repne; acha primeiro Not_Found de jnz de partida

o ah de mov, 9; funcionam 9 - dx de mov de barbante de exposição,
COMPENSAR Message6; ds: pontas de dx a 21h de int de mensagem; chamado
dos função

jmp Lodsb_Example

Not_Found: ah de mov, 9; funcionam 9 - dx de mov de barbante de
exposição, COMPENSAR Message7; ds: pontas de dx a 21h de int de mensagem;
chamado dos função

Lodsb_Example: ah de mov, 9; funcionam 9 - dx de mov de barbante de
exposição, COMPENSAR NewLine; ds: pontas de dx a 21h de int de mensagem;
chamado dos função

o cx de mov, 17; comprimento de si de mov de barbante, COMPENSAR
Mensagem; DS: SI - endereço de bh de xor de barbante, bh; página videa -
0 ah,0Eh de mov; 0Eh de função - escreve NextChar de caráter: lodsb; AL =
próximo caráter em 10h de int de barbante; NextChar de volta de serviço
de BIOS de chamado

ax,4C00h de mov; retorno a DOS 21h de int

. dados

O equ de CR 13 equ de LF 10

O db String1 "Isto é uma barbante! \$" db String2 18 dup (0)

O db Diff1 "Esta barbante é quase o mesmo como Diff2\$" db Diff2 "Esta
barbante é quase o mesmo como Diff1\$"

O db Equal1 "As barbantes são iguais\$" db Equal2 "As barbantes não são
semelhante\$"

SearchString db "1293ijdkfjiu938uHello983fjkijsi98934\$"

O db de mensagem "Isto é uma mensagem"

O db Message1 "programa de exemplo de instruções de Barbante que usa. \$"
CR,LF,"String1 de db Message2 está agora: \$" CR,LF,"String2 de db
Message3 está agora: \$" CR de db Message4, LF," Barbantes estão iguais!
\$" CR de db Message5, LF," Barbantes não são semelhante! \$" CR de db
Message6, LF," Caráter foi achado. \$" CR de db Message7, LF," Caráter nao
foi achado. \$"

NewLine db CR,LF,"\$"

o fim

COMO SABER O DOS ----- DE VERSÃO--.

Em muitos programas é necessário saber o que o DOS versão é. Isto podia
ser porque estão usando um DOS função que necessita a revisão ser sobre
um certo nível.

Primeiramente este método simplesmente sabe o que a versão é.

ah,30h de mov; 30h de função - recebe 21h de int de versão de MS-DOS;
chamado DOS função

Esta função retorna o número importante de versão em AL e o número menor
de versão em AH. Por exemplo se era versão 4.01, AL seriam 4 e AH seriam
01. O problema é que se em DOS 5 e SETVER mais alto podem mudar a versão
que é retornada. O meio de ficar redondo isto é usar este método.

ah,33h de mov; 33h de função - real DOS al,06h de mov de versão; 21h de
int 06h de subfunction; chamado interrompe 21h

Isto só trabalhará em DOS versão 5 e acima então necessita verificar usar
o método anterior. Isto retornará a versão real de DOS mesmo que SETVER
mudou a versão. Isto retorna a versão importante em BL e a versão menor
em BH.

MÚLTIPLOS EMPURRÕES E ----- DE
ESTOUROS--.

Pode empurrar e poder estourar mais de um registro numa linha em TASM e
A86. Isto faz seu código mais fácil entender.

o dx de cx de bx de machado de empurrão; poupa machado de bx de cx de dx de estouro de registros; restaura registros

Quando TASM (ou A86) compila estas linhas que traduz ele em empurrões separados uns estouros. Este meio somente poupa-o tempo bater e faz mais fácil entender.

A nota: fazer estas linhas compilar em A86 que você necessita por vírgulas (,) em entre os registros.

O PUSHA/PUSHAD E ----- DE INSTRUÇÕES DE POPA/POPAD--.

PUSHA é uma instrução útil que empurra todo propósito geral registra sobre a pilha. Realiza o mesmo como o seguinte:

o temp = di de empurrão de si de empurrão de bp de empurrão de temp de empurrão de bx de empurrão de dx de empurrão de cx de empurrão de machado de empurrão de SP

A vantagem principal é que ele menos está batendo, uma instrução menor e é um lote mais rápido. POPA faz o inverso e estoura estes registros fora a pilha. PUSHAD e POPAD fazem o mesmo mas com o ESP de 32 bits de registros, EAX, ECX, EDX, EBX, EBP, ESI e EDI.

AS MUDANÇAS QUE USAM PARA MULTIPLICATION MAIS RÁPIDO E ----- DE DIVISÃO--.

O MUL que usa e DIV é muito lentos e só deveu ser usado quando não corre é necessitado. Para multiplication mais rápido e divisão você pode mudar números à esquerda ou direito ou posições mais binárias. Cada mudança é a um poder de 2. Isto é o mesmo como o << and >> Operadores em C. Há quatro meios diferentes de mudar números qualquer um deixaram ou direito posição binária.

Unsigned de SHL múltiplo por dois Unsigned de SHR divide por dois SAR Assinou divide por dois SAL mesmo como SHL

A sintaxe para todos quatro é a mesmo.

A sintaxe: operand1,operand2 de SHL

A nota: Os 8086 nao podem ter o valor de opperand2 outro que 1. 286/386

nao pode ter operand2 mais altos que 31.

----- DE VOLTAS--.

A Volta que usa é um melhor meio de fazer uma volta JMP então usando. Coloca a quantia de vezes que você quer ele a volta no registro de CX e cada tempo alcança a declaração de volta ele (CX-1 DE CX de decrements) e então faz um pulo curto à etiqueta indicado. Um meio curto de pulo que pode só 128 bytes antes de ou 127 bytes depois da instrução de VOLTA.

A sintaxe: cx de mov, 100 ;100 vezes a Etiqueta de volta: . . . A Etiqueta de volta:; CX de decrement e volta Marcar

Isto é exatamente o mesmo como o seguinte pedaço de código sem usar volta:

o cx de mov, 100 ;100 vezes a Etiqueta de volta: cx de dec; CX = CX-1
Etiqueta de jnz; continua até que feito

Qual pensa é mais fácil entender? O DEC/JNZ que usa é mais rápido em 486's e acima e está útil como você nao tem que usar CX.

Isto trabalha porque JNZ pulará se a zero bandeira nao foi posta. Por CX a 0 porá esta bandeira.

COMO USAR UM ----- DE DEBUGGER--.

Isto é um bom tempo de usar um debugger saber o que seu programa realmente está fazendo. Estou indo demonstrar como usar Debugger de Turbo verificar o que o programa realmente está fazendo. Primeiro necessitamos um programa que nós podemos olhar em.

O alistamento 12: DEPURA.ASM

; programa de exemplo demonstrar como usar um debugger

. modelo minúsculo. 100h de org de código começa:

o machado de empurrão; poupa valor de bx de empurrão de machado; poupa valor de cx de empurrão de bx; poupa valor de cx

o machado de mov, 10; primeiro parâmetro é 10 bx de mov, 20; segundo parâmetro é 20 cx de mov, 3; terceiro parâmetro é 3

Chame ChangeNumbers; procedimento de chamado

o cx de estouro; restaura bx de estouro de cx; restaura machado de estouro de bx; restaura dx

ax,4C00h de mov; saída a dos 21h de int

PROC de ChangeNumbers adiciona machado, bx; adiciona número em bx a cx de mul de machado; multiplica machado por dx de mov de cx, machado; resposta de retorno em ENDP de ChangeNumbers de ret de dx

o fim começa

Agora compile-o a um. arquivo de COM e então tipo:

o nome de td de arquivo

Debugger de Turbo então carrega. Pode ver as instruções que compõem seus programas, por exemplo as primeiras poucas linhas deste programa é mostrada como:

o cs: 0000 50 cs de machado de empurrão: 0001 53 cs de bx de empurrão: 0002 51 cx de empurrão

Algumas teclas úteis para Debugger de Turbo:

F7 de Janela de Tamanho F5 Próximo Passo F4 DE ALT de Corrida F9 de Instrução para trás

Os números que são ocupados os registros são diferentes que os que no código de fonte porque são representados em sua forma de hex (baseia 16) como isto é a base fácil de converter a e de binário e que é mais fácil entender que binário também.

Na esquerda desta exposição há uma caixa que mostra o conteúdo dos registros. Em desta vez todos os registros principais estão vazios. Agora F7 de prensa este meio que a primeira linha do programa está corrida. Como a primeira linha empurrou o registro de MACHADO na pilha, você pode ver que o pointer de pilha (SP) mudou. Aperte F7 até que a linha que contem ax,000A de mov é destacada. Agora aperte-o novamente. Agora se olha na caixa que contem o conteúdo dos registros você pode ver aquele MACHADO contem A. Aperte-o novamente e BX agora contem 14, aperta-o novamente e CX contem 3. Agora se aperta F7 novamente você pode ver

aquele MACHADO agora contem 1E que está UM + 14. Aperte-o novamente e agora MACHADO contem 1E 5A multiplicou por 3. F7 de prensa novamente e você verá que DX agora também contem 5A. Aperte-o três mais vezes e você podem ver que CX, BX e MACHADO são todas costas fixas a seus valores originais de zero.

MAIS PRODUÇÃO EM ----- DE MODOS DE TEXTO--.

Estou indo cobrir mais meios de texto de outputting em modos de texto. Este primeiro programa é um exemplo de como mover o cursor e exibir uma barba de texto onde você quer ir.

O alistamento 12: TEXT1.ASM

. modelo minúsculo. 100h de org de código começa: dh de mov, 12; dl de mov de col de cursor, 32; ah,02h de mov de fila de cursor; move cursor ao bh direito de xor de lugar, bh; página vinda 0 10h de int; serviço de bios de chamado

o dx de mov, COMPENSAR Texto; DS: DX aponta a ah de mov de mensagem, 9; funciona 9 - 21h de int de barba de exposição; chamado dos serviço

ax,4C00h de mov; saída a dos 21h de int

DB de texto "Isto é algum texto\$"

Isto próximo exemplo demonstra como escrever à tela que usa o 40h de função de arquivo de interromper 21h.

O alistamento 13: TEXT2.ASM

o fim começa. modelo pequeno. pilha. machado de mov de código, @dados; armam ds como o segmento para ds de mov de dados, machado; põe isto em ds

ah,40h de mov; 40h de função - escreve bx de mov de arquivo, 1; cabo = 1 (tela) cx de mov, 17; comprimento de dx de mov de barba, COMPENSAR Texto; DS: pontas de DX a 21h de int de barba; chamado DOS serviço

ax,4C00h de mov; termina 21h de int de programa

. dados

DB de texto "Isto é algum texto"

o fim

O próximo programa mostra como armar e chamar 13h de função de interromper 10h - escreve barbante. Isto tem as vantagens de ser capaz de escrever uma barbante em qualquer lugar na tela numa cor especificada mas é duro de armar.

O alistamento 14: TEXT3.ASM

. modelo pequeno. pilha. machado de mov de código, @dados; armam ds como o segmento para es de mov de dados, machado; põe isto em bp de mov de es, COMPENSAR Texto; ES: pontas de BP a mensagem

ah,13h de mov; funciona 13 - escreve al,01h de mov de barbante; attrib em bl, move bh de xor de cursor, bh; página videa 0 bl de mov, 5; atributo - cx de mov de magenta, 17; comprimento de dh de mov de barbante, 5; fila por dl de mov de barbante, 5; coluna por 10h de int de barbante; serviço de BIOS de chamado

ax,4C00h de mov; retorno a DOS 21h de int

. dados

DB de texto "Isto é algum texto" fim

O próximo programa demonstra como escrever ao tela usar stosw de rep por o escrito em memória videa.

O alistamento 15: TEXT4.ASM

. modelo pequeno. pilha. ax,0B800h de mov de código; segmento de memória videa es intermediária de mov, machado; põem isto em di de xor de es, di; aclara di, ES: pontas de DI a ah video de mov de memória, 4; atributo - al vermelho de mov," G"; caráter por aí cx de mov, 4000; quantia de vezes pô-lo aí cld; direção - stosw dianteiro de rep [sbrk] ; caráter de produção em ES: [DI]

ax,4C00h de mov; retorno a DOS 21h de int

o fim

O próximo programa demonstra como escrever uma barbante a memória vídeo.

O alíamento 15: DIRECTWR.ASM

; escreve uma barbante dirigir a memória vídeo

. modelo pequeno. pilha. machado de mov de código, @ds de mov de dados, machado

ax,0B800h de mov; segmento de memória vídeo es intermediária de mov, machado; põem isto em es

o ah de mov, 3; atributo - cx de mov de cyan, 17; comprimento de barbante imprimir si de mov, COMPENSAR Texto; DX: pontas de SI a di de xor de barbante, di

Wr_Char: lods; põe próximo carácter em es de mov de al: [di],al; carácter de produção a di vídeo de inc de memória; move ao longo de a próximo es de mov de coluna: [di],ah; atributo de produção a Wr_Char vídeo de volta de di de inc de memória; volta até feito

ax,4C00h de mov; retorno a DOS 21h de int

. dados

DB de texto "Isto é algum texto"

o fim

É deixado como um exercício ao leitor modificá-lo de modo que só um escreve é feito a memória vídeo.

----- 13h de MODO--.

13h de modo está só disponível em VGA, cartões de MCGA e acima. A razão eu estou conversando sobre este cartão é que é muito fácil de usar para gráfico por causa de como a memória é organizada.

PRIMEIRO VERIFICAR AQUELE 13h de MODO É POSSÍVEL

-----.

Seria gentil contar o operador se computador do seu/seu não pudesse

apoiar 13h de modo em vez de crashing justo seu computador sem aviso.
Isto é como é feito.

O alistamento 16: CHECK13.ASM

. modelo pequeno. pilha. dados

O db de NoSupport "13h de Modo nao é apoiado neste computador." o db,"
necessita qualquer um um MCGA ou VGA video" ,"card/monitor. de db\$" db
Apoiado "13h de Modo é apoiado neste computador. \$"

. código

o machado de mov, @dados; armam DS apontar a ds de mov de segmento de
dados, machado; machado de uso

Check_Mode_13h de chamado; verificar se 13h de modo é possível Erro de
jc; se cf = 1 há um erro

o ah de mov, 9; funcionam 9 - dx de mov de barbante de exposição,
COMPENSA Apoiado; DS: pontas de DX a 21h de int de mensagem; chamado DOS
serviço

To_DOS de jmp; saída a DOS

O erro: ah de mov, 9; funcionam 9 - dx de mov de barbante de exposição,
COMPENSAR NoSupport; DS: pontas de DX a 21h de int de mensagem; chamado
DOS serviço

To_DOS: ax,4C00h de mov; saída a DOS 21h de int

PROC Check_Mode_13h; Retornos: CF = 1 13h de Modo nao possível

Ax,1A00h de mov; Pede info video para 10h de int de VGA; Recebe al,1Ah de
cmp de Código de Combinação de Exposição; VGA ou presente de MCGA É?
Mode_13h_OK de je; 13h de modo é apoiado stc; 13h de modo nao é apoiado
CF = 1

Mode_13h_OK: ret

Check_Mode_13h ENDP

o fim

Uso justo que isto verificar se 13h de modo é apoiado no começo de seu programa assegura-se que pode entrar naquele modo.

POR O ----- VIDEO DE MODO--.

É muito simples por o modo. Isto é como é feito.

ax,13h de mov; 10h fixo de int 13h de modo; serviço de BIOS de chamado

Veja uma que nós estamos em 13h de modo e acabou o que nós estamos fazendo necessitamos a nós necessita pô-lo ao modo video que estava em previamente. Isto é feito em duas etapas. Primeiramente necessitamos poupar o modo video e então reset ele àquele modo.

VideoMode db? ah,0Fh de mov; 0Fh de função - fica atual 10h de int de modo; Bios VideoMode video de mov de chamado de serviço, al; poupar modo atual

; código de programa aqui

o al de mov, VideoMode; ah video, prévio fixo de xor de modo, ah; aclaram ah - 10h fixo de int de modo; ax,4C00h de mov de serviço de bios de chamado; saída a dos 21h de int; chamado dos função

Agora que podemos fazer-nos parte de 13h de modo deixa fazer algo. Primeiramente deixa pôe algum pixels na tela.

0Ch de função: Escreve Pixel Gráfico

Isto faz um ponto de cor na tela no especificado gráfico coordena.

A ENTRADA: AH = AL 0Ch = Cor do CX de ponto = coluna de Tela (x coordena)
DX = fila de Tela (y coordena)

A PRODUÇÃO: Nada exceto pixel em tela.

A nota: Esta função executa exclusivo OU (XOR) com o novo valor de cor e o contexto atual do pixel de bit 7 de AL é posto.

Este programa demonstra como conspirar pixels. Deve conspirar quatro pixels vermelho no meio da tela.

O alistamento 17: PIXINT.ASM

; exemplo de conspirar pixels em modo 13 serviços de bios que usam -; 10h de INT

. modelo minúsculo. 100h de org de código

comece: machado de mov, 13; modo = 10h de int 13h; serviço de bios de chamado

ah,0Ch de mov; al de mov 0Ch de função, 4; cor 4 - cx vermelho de mov, 160; posição de x = 160 dx de mov, 100; posição de y = 100 10h de int; serviço de BIOS de chamado

o dx de inc; pixel de trama para 10h baixo de int; cx de inc de serviço de BIOS de chamado; pixel de trama a 10h direito de int; dx de dec de serviço de BIOS de chamado; pixel de trama para 10h de int de cima; serviço de BIOS de chamado

o machado de xor, machado; 00h de função - recebe um 16h chave de int; machado de mov de serviço de BIOS de chamado, 3; modo = 3 10h de int; serviço de BIOS de chamado

ax,4C00h de mov; saída a DOS 21h de int

o fim começa

ALGUM ----- DE OPTIMIZATIONS--.

Este método não é muito rápido e nós podíamos fazer um lote mais rápido. Como? Por escrever dirija a memória videa. Isto é feito facilmente.

O segmento de VGA é 0A000h. Elaborar onde cada pixel vai-o usar esta fórmula simples receber o compensar.

Compensar = X + (x de Y 320)

Todo que nós fazemos é por um número nesta localidade e está agora um pixel na tela. O número é o que cor que é.

Há duas instruções que podemos usar para por um pixel na tela, primeiramente nós podíamos usar stosb por o valor em AL a ES: DI ou podemos usar uma nova forma da instrução de MOV como isto:

o es de mov: cor [di],

Qual devemos usar? Quando estamos indo escrever pixels à tela que nós necessitamos fazer então tão rápido quanto é possível.

Pentium de instrução 486 386 286 86

STOSB 3 5 4 3 11 AL DE MOV a SEG: FORA 1 1 4 3 10

Se usa o método de MOV você pode necessitar a DI de incremento (qual STOSB faz).

[põe instrução de pixel]

Se tivemos um programa que usou sprites que necessita ser continuamente tirar, apagado e então redraw você terá problemas com flicker. Evitar este pode usar um 'memória dupla intermediária'. Isto é outra parte de memória que você escreve a e então copia toda a informação sobre a tela.