Tutorial Assembly Parte 3: O ARRAY
By Mr. Maggots29A - Data: 06/01/2006

Nota do Autor:

E então Ano Novo e vida velha não tem nada melhor hein?
Claro que tem, o velho e bom Assembly codado nos primórdios 16 bits, sem contar que ainda compilado como .COM ☺. Você deve estar achando que estou tirando um sarro não é? E não é que estou mesmo! Mas calma eu sei que você quer mais eu sei que você não vê a hora de codar seu primeiro game em modo 13h com um visual bem no estilo Wolfstein 3D e impressionar seus amigos! Mas isso requer muita pratica e conhecimento, ou você acha que John Carmack (Criador do doom, quake e etc) estava atoa em casa e resolveu fazer um jogo do nada? Ele estudou muito e provavelmente deve ter começado com um Pong ou talvez com game texto que veremos nesse tutorial. Smile, You're on the Candid Camera! ☺

O Passado Não muito Distante:

Diferente da demora (leia-se meses) entre o primeiro e o segundo tutorial, dessa vez foi coisa de semanas, bem estou tentando melhorar com o tempo! Mas espero que os tópicos anteriores estejam fresco na sua mente.

No segundo tutorial você viu:

- - LABEL
- - FLAGS
- - COMPARAÇÃO
- - JUMP
- - A TABELA ASCII
- - PILHA
- - PUSH POP

E claro os mini programas para entender o uso de cada mnemônico!

Chega de papo furado vamos conhecer novas instruções, opcodes e afins...

A VARIÁVEL

A variável é um espaço na memória onde você coloca dados, e existem três “tipos básicos” de variáveis:

DB	DATA BYTE	1 Byte
DW	DATA WORD	2 Bytes
DD	DATA DOUBLE	4 Bytes

A escolha de cada tipo de dado vai de acordo com a necessidade do programador. Entretanto não veremos o tipo double por enquanto.

A sintaxe abaixo mostra a declaração de uma variável:

NOME_DA_VARIÁVEL TIPO VALOR

Viu como é fácil? Vamos ver isso na prática:

Exemplo de Variável:

```
org 100h

START:
        mov ah,02h
        mov dl,[MyVar]        ; Carrega o valor de MyVar em dl
        int 21h
        ret

MyVar db "A"
```

Quando faço o `mov dl,[MyVar]` estou trabalhando com Bytes. O que acontece se eu tiver uma carreira mais extensa, por exemplo:

```
org 100h

START:
        mov ah,02h
        mov dl,[MyVar]        ; Carrega o valor de MyVar em dl
        int 21h
        ret

MyVar db "ABC"
```

Como a minha instrução `mov ah,02h` da `int 21h` só compreende 1 Byte, então apenas "A" será impresso na tela do computador, mas se eu fizer diferente:

```
org 100h

START:
        mov ah,02h
        mov dl,[MyVar+1]      ; Carrega o conteúdo de MyVar + 1 que é o valor B
        int 21h
        ret

MyVar db "ABC"
```

O que acontece aqui é muito simples, é inicio de uma **STRING** (Que veremos mais tarde). O `MyVar` é como se fosse uma Label/Etiqueta para uma posição de memória X. Exemplo:

MyVar	→	**A**	**B**	**C**
Memória	→	1000	1001	1002

Então se temos a posição: `[MyVar] = A` (Endereço 1000), fazendo `[MyVar+1] = B` (Endereço 1001).

Está ficando legal não é? Agora vamos ver um exemplo, onde é impresso uma série de caracteres através do Loop. Exemplo:

```
org 100h

START:
        mov bx,MyVar          ; Sem o uso dos colchetes [] esse mov passa o endereço da
                              ; memória de MyVar para bx.
MAIN_LOOP:
        mov ah,02h
        mov dl,[bx]           ; Aqui estou movendo o conteúdo para o qual bx aponta ou seja
                              ; o conteúdo da variável MyVar para o registrador dl.
        int 21h
        inc bx                ; Agora incremento o endereço de bx em 1 unidade.
        cmp dl,'0'            ; Comparação entre dl e o caracter '0'.
        jne MAIN_LOOP         ; Salta se dl e '0' não for iguais.
        ret                   ; Sai do programa quando a condição for satisfeita.

MyVar db "1234567890"         ; Minha variável.
```

DICAS DE MOVIMENTOS:

É preciso prestar atenção sobe os tipos de MOV que podem ser feitos com as variáveis:

```
Mov Reg,Var       ; Está sendo movido o endereço da Variável para um Registrador, nesse caso o
                  ; registrador sempre será AX,BX,CX,DX ou seja 16 bits se forem programados
                  ; nessa plataforma.

Mov Reg,[Var]     ; Está sendo movido o conteúdo da Variável para um Registrador, diferente do caso
                  ; acima o registrador deve ter o mesmo tamanho do Registrador.

Mov Var_1,Var_2   ; Movimentos entre variáveis são proibidos, o valor tem que passar antes por um
                  ; registrador.
```

A VAR SEM IDENTIDADE:

Não não e não a Var não é uma garota louca que mora perto da minha casa. A Var continua sendo a velha e boa Variável! ☺ Essas piadas matam né?

O.k. se você não gostou não tem problema, mas então você pode criar uma variável sem um valor definido, vamos ver como fica:

```
MyVar       db      ?

MyVar_2     dw      ?
```

Isso é muito útil quando você está usando Assembly com outra linguagem de Alto Nível!

VAR e DUP

O Dup você usa para definir o mesmo valor várias vezes, ou para alocar um tamanho limite na memória!

```
MyVar       db      3 dup (A)       ; Com isso MyVar = AAA.

MyVar_2     db      5 dup (?)       ; É uma referencia quando você quer alocar um espaço na
                                    ; memória. Nesse caso temos 5 bytes alocados.
```

Veremos o uso de DUP mais a frente, quero dar uma pausa com as variáveis e apresenta-los algo quase igual só que diferente. ☺

O PODER DA CADEIA DE CARACTERES:

A cadeia é lugar ruim e perigoso, mas a cadeia de caracteres é diferente é uma cadeia onde só existem bytes das melhores espécies... humm assim eu espero! ☺

Uma cadeia de caracteres também é conhecida como String e nada mais é que uma seqüência de caracteres, e seu tamanho consiste na quantidade de caracteres ou bytes que contem essa cadeia! EX:

MinhaString = “OLA PESSOAL”

MinhaString: É um nome genérico da minha variável, apenas informa onde estão as caracteres na memória.

“OLA PESSOAL”: Essa é a seqüência de caracteres, e lembrando que cada caractere é igual a 1 byte, então temos 11 bytes gastos nessa frase. E lembre-se o espaço em branco também eqüivale a 1 byte.

Note que seu uso é espetacular para qualquer tipo de programa onde há a comunicação entre máquina e usuário. Pois imagina o seguinte, você tem que criar um programa que da as boas vindas ao usuário, já pensou ter que imprimir caractere por caracter até satisfazer as boas vindas? Seria extremamente irritante, então vamos lá mande os Bytes para a Cadeia e deixe eles se rebelarem! ☺

MINHA PRIMEIRA VEZ NA CADEIA:

A primeira coisa que você precisa fazer quando se entra na cadeia, é fazer amizades e preparar o terreno.

Abra o FASMW:

```
org 100h                                ; Sim é um programa .COM;

        mov ah,09h                      ; ah = 09h quer dizer que estamos trabalhando com String;
        mov dx,MinhaString              ; dx = MinhaString É a copia do endereço para dx;
        int 21h                         ; Lê ah e imprime o conteúdo apontado pelo endereço em dx;
        ret                             ; Sai do programa;

MinhaString     db      "Ola Mundo!",24h  ; Nossa String ou Cadeia de Caracteres;
```

Pois bem ai esta o programa que irá imprimir o mais que famoso “Ola Mundo!”, mas é lógicos que muitas coisas ainda precisam ser explicadas, como o:

24h depois da String compreende o EOL = End-Of-Line ou Término de Linha.

Note como foi simples, fácil e limpo entrar e sai dessa cadeia!

Legal? Agora vamos ao nosso primeiro game:

GAME: ADIVINHA NÚMERO:

A mecânica desse game é muito simples, o usuário tem que acertar um número imposto pelo computador. No nosso caso o número será pré definido antes do inicio do game, nos próximos capítulos porem será necessário criar uma rotina de números aleatórios.

Bem eu fiz 2 versões desse jogo, a primeira bem simples não detecta muitos erros, não diferencia números de letras, ou seja quando pedir para o usuário digitar um número, o programa entenderá que A = ‘65’ ASCII é um número. O legal dessa versão é você entender o espirito da programação e do jogo, a segunda versão está cheia de aprimoramentos, tem opção de sair quando quiser, diferencia números de letras, e etc. Aqui estarei colocando apenas a primeira versão, já a segunda versão está para download junto com os sources desse tutorial.

Note que é um jogo bem simples, o esquema é comparar e colocar uma mensagem na tela. O número pré definido é o número 5, se o usuário digitar qualquer número menor que 5 ou seja 0,1,2,3,4 deverá ser impresso a mensagem que o número foi menor. Se o usuário digitar um número maior que 5, sendo 6,7,8,9 deverá ser impresso que o número digitado foi maior que o escolhido. E quando o usuário digitar 5, imprimir a mensagem de acerto e sair do game.

O legal é que com o esquema base você poderá começar aprimorar o game, colocando uma opção de saída quando for pressionado a tecla ESC, entre outras coisas como fiz na segunda versão. O ideal na minha opinião é você tentar aprimorar a base que você verá abaixo, sem olhar a segunda versão.

Então vamos ver o joguinho.

JOGO ADIVINHA NÚMERO V.1:

```
org 100h

START:
        mov ah,09h
        mov dx,Msg_Inicial
        int 21h

MAIN_LOOP:
        mov ah,09h
        mov dx,Msg_Texto
        int 21h

        mov ah,00h
        int 16h

        cmp al,'5'
        je ACERTOU
        jb MENOR
        ja MAIOR

ACERTOU:
        mov ah,09h
        mov dx,Msg_Acertou
        int 21h
        ret

MENOR:
        mov ah,09h
        mov dx,Msg_Menor
        int 21h
        jmp MAIN_LOOP

MAIOR:
        mov ah,09h
        mov dx,Msg_Maior
        int 21h
        jmp MAIN_LOOP

Msg_Inicial db 13,10,'Adivinhe o Numero - Apenas digite numeros entre 0 a 9!',13,10,24h
Msg_Texto   db 13,10,'DIGITE UM NUMERO:',24h
Msg_Menor   db 13,10,'VOCE DIGITOU UM NUMERO MENOR!',24h
Msg_Maior   db 13,10,'VOCE DIGITOU UM NUMERO MAIOR!',24h
Msg_Acertou db 13,10,'PARABENS VOCE ACERTOU!',24h
```

Conclusão:

Esse jogo não requer muita explicação por que ele é praticamente auto explicável, no inicio do código é logo mostrado uma mensagem com um explicação simples do jogo, em seguida vem o Main_Loop onde praticamente todo o jogo se resume, depois é feito a leitura do teclado com `mov ah,00h` e `int 16h` e após o digito do usuário vem comparação. Note os jumps aninhados, isso é normal, pois com uma comparação você pode tomar diversos caminhos diferentes.

UM DETALHE - 13,10,

Note que em todas as nossas mensagens temos os códigos 13,10,! Já matou a charada? Se você decorou a tabela ASCII sabe que é 13 é um alimentador de linha, e 10 é um retorno de carro. Ou seja 13 pula uma linha e 10 coloca o cursor no inicio dessa linha.

Então vocês já sabem que a lição de casa é aprimorar esse jogo do Adivinha Número.

O QUE É ISSO?

Pessoal eu estava navegando na Internet quando eu comecei há ver pessoas falando de um site nacional sobre Programação Assembly. Eu lembro que quando estava procurando por tutoriais Assembly eu já tinha visto esse tal site que as pessoas comentavam, e hoje resolvi rever o site e ver como ele estava. Foi então que comecei a lembrar que uma das coisas que eu não gostava no site eram os códigos, eu tentava entender aqueles códigos mas não era possível, parecia ser uma coisa psicótica, cheguei a pensar que o problema era comigo. Mas hoje ao visita-lo descobri que o problema estavam nos códigos! AINDA BEM. ☺ Pois bem nesse site tem um programa descrito como:

Melhorando o "Escreva alguma coisa" <- Não é brincadeira esse é o titulo de verdade!

```
[ORG 0x100]

inicio:
   mov     ah, 9                   ; Serviço de impressão da INT 21
   mov     dx, msg1                ; Imprime "Escreva alguma coisa: "
   int     0x21

tecla:
   mov     ah, 0x1                 ; Serviço de rastrear uma tecla
   int     0x21

   cmp     al,13                   ; se for Enter
   je      termina                 ; salte para termina e põe caracteres na tela
   cmp     al, 27                  ; se foe Esc
   je      fecha                   ; termina o programa
   mov     [data+bx], al           ; põe caracter da tecla no buffer
   mov     bl, [count]             ; põe número já teclado em BL
   inc     bx                      ; incrementa número de caracteres
   mov     [count], bl             ; atualiza count
   cmp     bl, 20                  ; compara número de caracteres com 20
   je      termina                 ; se forem 20, termina
   jmp     tecla                   ; senão volta a esperar tecla
termina:
   mov     al, '$'                 ; Põe caracter terminador '$' em AL
   mov     bl, [count]             ; Põe número de teclas digitadas em BL
   mov     bh, 0                   ; Zera o byte alto de BX
   mov     [data+bx], al           ; Adiciona '$' no final do buffer

   mov     ah, 9                   ; Imprime "Você escreveu: "
   mov     dx, msg2                ;
   int     0x21                    ;

   mov     ah, 9                   ; Imprime o conteúdo do buffer
   mov     dx, data
   int     0x21

   xor     bx, bx                  ; Zera o registrador BX (o mesmo que mov bx,0)
   mov     bl, 20                  ; Máximo de caracteres (20)
   xor     ax, ax                  ; Zera o registrador AX
   mov     [count], al             ; Zera o contador
limpa:
   mov     [data+bx], al           ; Põe 0 na posição início do buffer + BL
   dec     bl                      ; Decrementa BL
   cmp     bl, 0                   ; Compara com 0
   ja      limpa                   ; Se for maior que 0 continua limpando o buffer

   jmp     inicio                  ; Se não, pede "Escreva alguma coisa: "

fecha:
   mov     ax, 0x4c00      ; Serviço para terminar programa
   int     0x21

; Os caracteres 13 e 10 forçam uma nova linha
msg1    db      13,10,13,10,'Escreva alguma coisa: $'
msg2    db      13,10,'Você escreveu: $'

; Este é o buffer de entrada com espaço para 20 bytes para caracteres
; e 1 byte para '$'
buf:
count   db      0
data    db      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

É incrível como até hoje não entendo o código desse cara. Vamos ao meu código que é mais claro e fácil de entender:

PROGRAMA FRASE:

```
org 100h

START:
        mov ah,09h
        mov dx,Prompt
        int 21h
MAIN:
        mov ah,00h
        int 16h

        cmp al,13
        je TEXTO
        cmp al,27
        je FIM
        cmp bx,20
        je TEXTO

        mov [Buffer+bx],al
        inc bx
        jmp MAIN
TEXTO:
        mov ah,09h
        mov dx,Frase
        int 21h

        mov [Buffer+bx],24h
        mov dx,Buffer
        int 21h

        cmp bx,0
        ja LIMPA_BUFFER
        je START
LIMPA_BUFFER:
        mov [Buffer+bx],0h
        dec bx
        cmp bx,0h
        jne LIMPA_BUFFER
        je START
FIM:
        ret

Prompt  db 13,10,"Ola Digite uma frase:",24h
Frase   db 13,10,"Voce Digitou: ",24h
Buffer  db 20 dup (?)
```

COMO DIRIA HOMER SIMPONS DUH... MAS NO NOSSO CASO DUP!

Bem ai está o uso do Dup para o qual eu falei, veja o meu Buffer com o do programa acima, note como Dup deixa o código mais limpo e organizado e..... NOSSA CHEGAMOS AO FIM DE MAIS UM TUTORIAL. Espero que você tenha gostado e não esqueça de estudar e claro se possível tentar coisas novas.

Eu: Por favor você trabalha com AMD?
VENDEDOR: Sim!
Eu: Você acha AMD um bom Processador?
VENDEDOR: Se eu trabalho com essa Marca é por que é coisa boa!
Eu: O.k. e você teria a placa mãe EPOX Modelo XXXXX
Vendedor: Humm cara não fica gastando dinheiro com placa-mãe, bota qualquer Asus vagabunda que dá conta do recado!
***OBS:** SERIO ELE FALOU ASSIM MESMO ASUS VAGABUNDA! VAI ENTENDER ESSE POVO.*

Stigma29a – By: Matheus – matheus_nab@hotmail.com