

Introdução à Software Básico: Introdução a Assembly IA-32

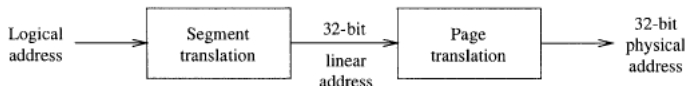
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade de Brasília

Sumário

- 1 Paginação
- 2 Tipos de Dados
- 3 Modos de Endereçamento

Modo Nativo - Protected Mode

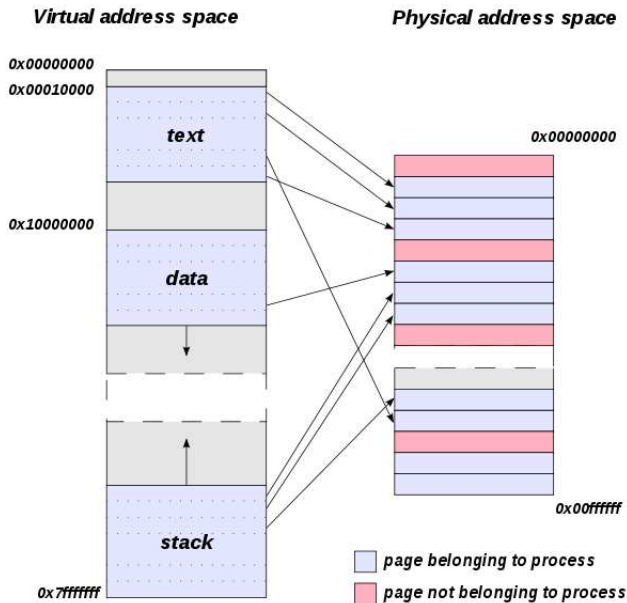
- Neste modo existe tanto paginação quanto segmentação
- Segmentação traduz endereços lógicos em endereços lineares.
- Paginação é utilizada para memória virtual
 - Transparentes para programas aplicativos.
 - Traduz endereços lineares em endereços físicos.



Paginação

- Mediante a paginação é possível restringir o acesso de memória para determinados aplicativos de forma que possam acessar somente uma seção da memória (pagina). Isto impede que um processo compartilhe/manipule a memória que esta sendo usada por outro.
- Além disso a paginação pode ser utilizada para criar memória virtual (por exemplo em HD).

Movimentação de Dados



Definição

- São valores numéricos armazenados em posições de memória e associados a um nome, equivalente ao endereço de memória onde esses valores estão armazenados.
- Os valores numéricos armazenados podem representar números (inteiros ou ponto flutuante) ou caracteres (códigos numéricos associados a letras)
- A quantidade de posições de memória (bytes) ocupadas por cada variável define a faixa de valores que essa variável pode assumir:
 - 1 *byte* = 8 *bits* $\rightarrow 0$ a $2^8 - 1$
 - 2 *bytes* = 16 *bits* $\rightarrow 0$ a $2^{16} - 1$
 - ...

Localização

- Podem estar nos registradores, e nesse caso não possuem nome, mas o nome do registrador é usado
 - `mov EAX,EBX`
- Uma variável pode estar em uma região de memória e seu endereço vai depender do modelo de acesso de memória escolhido. A variável então possui um nome:
 - `response DB ' Y'` ; allocates a byte, initializes to Y
 - `mov AL,[response]` ; copies response into register AL
 - `total RESB 1` ;allocates one byte
 - `mov [total],56` ;56 is written in total

Definição

- Constantes são valores mantidos fixos pelo compilador e não podem ser alterados após inicialização.
- Existem três maneiras de definir constantes em C.
 - Usando a diretiva de pré-processamento **#define**
 - `#define pi 3.1415`
 - Usando a palavra-chave **const** ao declarar uma variável
 - `const int id_no = 12345;`
 - A terceira maneira é usando enumerações. Uma enumeração define um conjunto de constantes.
 - `enum day`
`{ saturday,sunday,monday=1,tuesday,`
`wednesday,thursday,friday};`

Constantes em Assembly

- NASM possui três diretivas: EQU, %assign e %define.
 - EQU: define constantes numéricas e não permite redefinições no mesmo programa
 - CR EQU ODH ;carriage-return character
 - %assign: define constantes numéricas e permite redefinições no mesmo programa
 - %define: define constantes numéricas e strings e permite redefinições no mesmo programa

Exemplos

```
NUM OF ROWS EQU 50
NUM OF COLS EQU 10
ARRAY SIZE EQU NUM OF ROWS * NUM OF COLS

;%assign permite redefinição
%assign i j+1
...
...
%assign i j+2

;%define pode ser usado para strings
%define XI [EBP+4]
...
...
%define XI [EBP+20]
```

Declaração de Variáveis em Assembly

- Basta reservar espaço para as variáveis com as diretivas do Montador:
 - Inicializadas, com DB,DW,DD,DQ,DT
 - Não inicializadas, com RESB, RESW, RESD, RESQ, REST
- A inicialização seria uma atribuição de valor à variável

Atribuição de Valores em Assembly

- Instrução mais simples de atribuição: MOV
- Formato:
 - `mov destino, fonte`
 - O valor do operando fonte é copiado para o operando destino.
- Destino pode ser um registrador ou posição de memória.
- Fonte pode ser um registrador, posição de memória ou valor imediato.

Assembly: Instrução MOV

- Formatos possíveis da instrução MOV

```
mov register,register  
mov register,immediate  
mov memory,immediate  
mov register,memory  
mov memory,register
```

Exemplos de uso da instrução

```
section .data  
response DB ' Y'  
table1 TIMES 20 DW 0  
name1 DB 'Jim Ray'  
section .text  
mov AL,[response]      ;Moving a Byte  
mov DX, [table1]       ;Moving a Word  
mov [response], 'N'    ;Moving a Byte  
mov [name1+4], 'K'     ;Moving a Byte
```

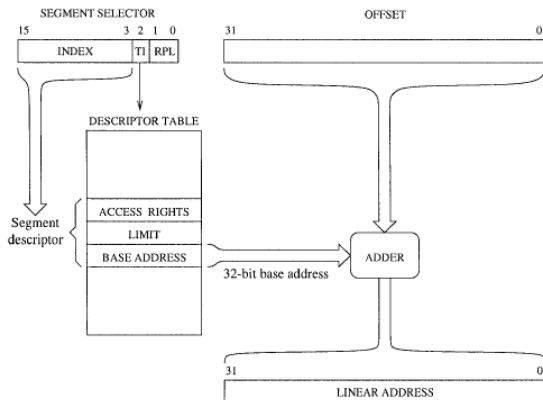
Conversão de Tipos na Atribuição em Assembly

- Como os tipos do Assembly são ligados ao seu tamanho, a instrução MOV converte os tipos de acordo com o tamanho do dado sendo atribuído
- O tamanho da instrução pode ser definido explicitamente:
 - `mov WORD [EBX],100` ;move Word
 - `mov BYTE [ESI],100` ;move Byte
- O tamanho dos dados na instrução MOV também pode ser definido pelo tamanho do registrador indicado na instrução
 - `mov AL,[response]` ;move byte
 - `mov DX, [table!]` ;move word
 - `mov ECX,10` ;Moving a DWord

Type specifier	Bytes addressed
BYTE	1
WORD	2
DWORD	4
QWORD	8
TBYTE	10

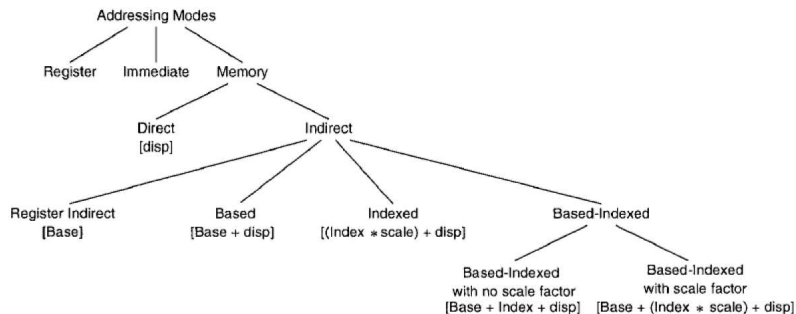
Escopo de Variáveis em Assembly

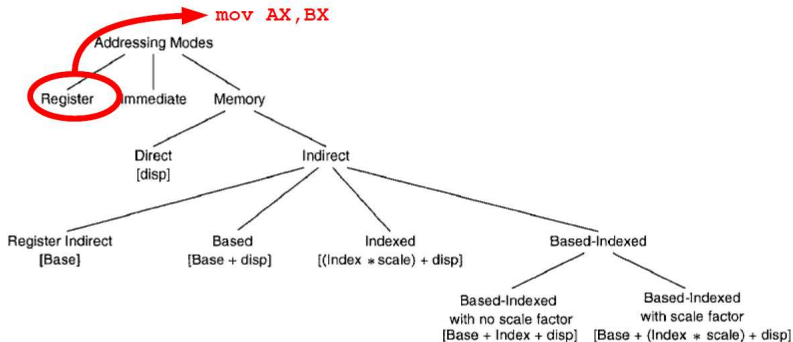
- A linguagem Assembly não fornece suporte para variáveis locais, ou mesmo parâmetros para funções
 - Uso da pilha para armazenar valores temporários, como parâmetros e variáveis locais
- O programa ou módulo que cria procedimentos deve salvar o contexto, criar o espaço na pilha e limpá-lo antes de retornar.
- O PROGRAMADOR EM ASSEMBLY DEVE CUIDAR DO ESCOPO DAS VARIÁVEIS LOCAIS!.

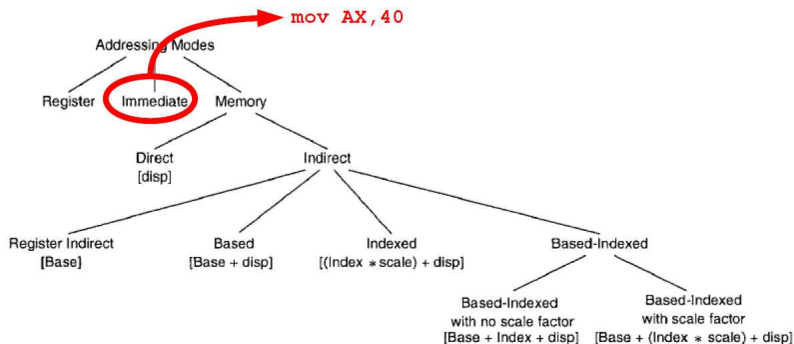


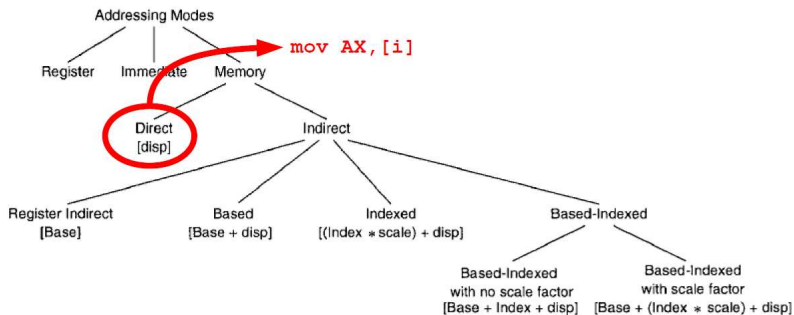
Segmentação e Ponteiros

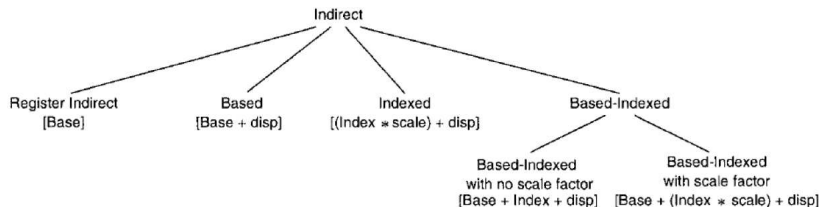
- Onde está a próxima instrução a executar?
 - CS: segment base; EIP: instruction offset
 - CS:EIP
- Onde está o topo da pilha?
 - push, pop: SS:ESP
- Onde estão as variáveis?
 - DS:xxx, onde xxx é o ponteiro para a variável
 - Os endereços das posições de memória são definidos pelos offsets dentro do segmento





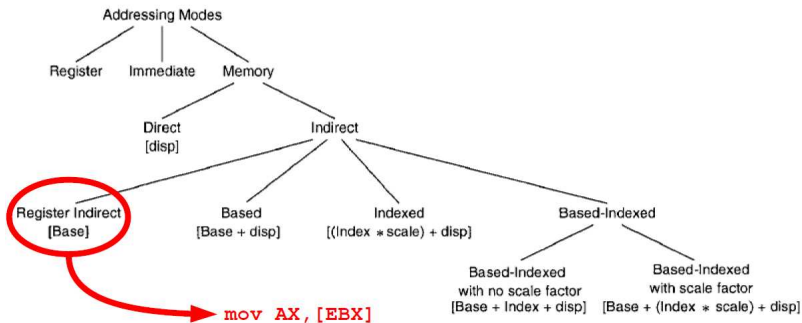






Modos de Endereçamento Indireto

- Permitem maior flexibilidade no acesso a estruturas de dados em alto nível, como Vetores, matrizes, registros, etc.



Segment + Base + (Index * Scale) + displacement

CS	EAX	EAX	1	No displacement
SS	EBX	EBX	2	8-bit displacement
DS	ECX	ECX	4	32-bit displacement
ES	EDX	EDX	8	
FS	ESI	ESI		
GS	EDI	EDI		
	EBP	EBP		
	ESP			

Based Addressing

- Um dos registradores fornece o endereço de um operando
- O endereço efetivo é calculado somando o conteúdo do registrador com o deslocamento fornecido como parte da instrução.
 - `mov ECX, [EAX+8]`
 - `add EAX, [EBX+4]`
- Interessante para acessar estruturas de dados tipo registros

Exemplo - parte 1

```
%define code      0
%define limit     4
%define registered 8
%define room      12
```

```
global _start
```

```
section .bss
```

```
struct    RESB 16
```

```
section .data
```

```
msg1 db "Entre com codigo do curso ",0
```

```
SIZE_MSG1 EQU  $-msg1
```

```
msg2 db "Entre com limite de alunos ",0
```

```
SIZE_MSG2 EQU  $-msg2
```

```
msg3 db "Entre com numero de alunos matriculados ",0
```

```
SIZE_MSG3 EQU  $-msg3
```

```
msg4 db "Entre com numero da sala ",0
```

```
SIZE_MSG4 EQU  $-msg4
```

```
section .bss
```

```
response resb 2
```

```
section .text
```

```
_start:
```

```
....
```

```
struct curso{
    int code;           4bytes
    int limit;          4bytes
    int registered;     4bytes
    int room;           4bytes
                        -----
                        16bytes
};
```

Exemplo - parte 2

```
...  
mov eax,4  
mov ebx,1  
mov ecx,msg1  
mov edx,SIZE_MSG1  
int 80h  
mov eax,3  
mov ebx,0  
mov ecx,response  
mov edx,2  
int 80h  
mov EBX,0  
mov BL,[response]  
sub BL,0x30  
mov EAX,struct  
mov DWORD [EAX+code],EBX    ;Código do Curso  
mov eax,4  
mov ebx,1  
mov ecx,msg2  
mov edx,SIZE_MSG2  
int 80h  
mov eax,3  
mov ebx,0  
mov ecx,response  
mov edx,2  
int 80h  
...
```

```
struct curso{  
    int code;           4bytes  
    int limit;          4bytes  
    int registered;     4bytes  
    int room;           4bytes  
};                      16bytes
```

Exemplo - parte 3

```
...  
    mov EBX,0  
    mov BL,[response]  
    sub BL,0x30  
    mov EAX,struct  
    mov DWORD [EAX+limit],EBX  
    mov eax,4  
    mov ebx,1  
    mov ecx,msg3  
    mov edx,SIZE_MSG3  
    int 80h  
    mov eax,3  
    mov ebx,0  
    mov ecx,response  
    mov edx,2  
    int 80h  
    mov EBX,0  
    mov BL,[response]  
    sub BL,0x30  
    mov EAX,struct  
    mov DWORD [EAX+registered],EBX ;Numero de alunos registrados  
    mov eax,4  
    mov ebx,1  
    mov ecx,msg4  
    mov edx,SIZE_MSG4  
...
```

```
struct curso{  
    int code;           4bytes  
    int limit;          4bytes  
    int registered;     4bytes  
    int room;           4bytes  
};                      16bytes
```

Exemplo - parte 4

```
...  
mov eax,3  
mov ebx,0  
mov ecx,response  
mov edx,2  
int 80h  
mov ebx,0  
mov BL,[response]  
sub BL,0x30  
mov EAX, struct  
mov DWORD [EAX+room],EBX ;Numero da sala  
mov eax,1  
mov ebx,0  
int 80h
```

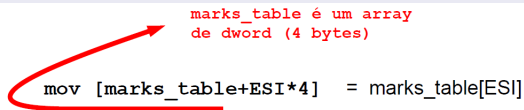
```
struct curso{  
    int code;           4bytes  
    int limit;          4bytes  
    int registered;     4bytes  
    int room;           4bytes  
};                      16bytes
```

Indexed Addressing

- Endereço efetivo é calculado: $(\text{Index} * \text{scale factor}) + \text{signed displacement}$
- **Displacement** aponta para o início de um array
- **Scale factor** indica tamanho dos elementos do array (2, 4 ou 8 bytes)
- **Index** indica um elemento dentro do array
- Prático para acessar arrays

marks_table é um array
de dword (4 bytes)

```
mov [marks_table+ESI*4] = marks_table[ESI]
```



Exemplo

```
global _start
section .data
test_marks      dd  90,50,70,94,81,40,67,55,60,73
NO_STUDENTS     EQU ($-test_marks)/4      ; number of students
sum_msg         db  'The sum of test marks is: ',0
SIZE_MSG        EQU  $-sum_msg

section .bss
sum             resd 1
sum_out         resb 10
size_sum_out    resb 1
section .text
_start:
    mov     ecx,NO_STUDENTS
    sub     eax,eax
    sub     esi,esi
add_loop:
    add     eax,[test_marks+ESI*4]
    inc     esi
    loop    add_loop
    mov     [sum],eax
    mov     eax,1
    mov     ebx,0
    int     80h
```

Based-Indexed Addressing

- Endereço efetivo é calculado: $\text{Base} + \text{Index} + \text{displacement}$
- Prático para acessar arrays bi-dimensionais
 - $A[i][j]$, onde A é um array $n \times m$, de elementos tamanho s

Based-Indexed Addressing

- Endereço efetivo é calculado: $\text{Base} + \text{Index} + \text{displacement}$
- Prático para acessar arrays bi-dimensionais
 - $A[i][j]$, onde A é um array $n \times m$, de elementos tamanho s
- Disposição do array na memória:
 - $A[0][0] \ A[0][1] \ A[0][2] \ A[0][3] \ \dots \ A[0][n-1] \ A[1][0] \ A[1][1] \ \dots \ A[1][n-1] \ A[2][0] \ A[3][0] \ \dots \ A[m-1][n-1]$

Based-Indexed Addressing

- Endereço efetivo é calculado: $\text{Base} + \text{Index} + \text{displacement}$
- Prático para acessar arrays bi-dimensionais
 - $A[i][j]$, onde A é um array $n \times m$
 - Base: início do array, A
 - Index: início da linha, $i * n * s$
 - displacement: índice de coluna, j

Exemplo

```
global _start
section .data
ROWS      EQU 10
COLUMNS  EQU 10
array1
    db 90,89,99,91,92,95,77,38,69,10
    db 79,66,70,60,55,68,70,60,77,10
    db 70,60,77,90,89,99,91,92,95,10
    db 60,55,68,79,66,70,60,55,68,10
    db 51,59,57,02,92,95,77,38,69,10
    db 79,66,70,60,55,68,70,60,77,10
    db 70,60,77,90,89,99,91,92,95,10
    db 60,55,68,79,66,70,60,55,68,10
    db 51,59,57,02,92,95,77,38,69,10
    db 79,66,70,60,55,68,70,60,77,10
array2
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
    db 1,2,3,4,5,6,7,8,9,10
Section .bss
array3 resb 100
```

```
section .text
; loop iteration count
_start:
    mov     ECX,ROWS
    mov     EBX,0
    mov     ESI,0
    mov     AH,0
    mov     DH,0
sum_loop:
    mov     AL,[array1+EBX+ESI]
    mov     DL,[array2+EBX+ESI]
    add     AL,DL
    mov     [array3+EBX+ESI],AL
    inc     ESI
    cmp     ESI,COLUMNS
    jb      sum_loop
    add     EBX,COLUMNS
    mov     ESI,0
    loop    sum_loop
    mov     eax,1
    mov     ebx,0
    int     80h
```

Próxima Aula

Modos de Endereçamento