Linguagens de Montagem

Modos de Endereçamento
Aula 08

Edmar André Bellorini

Ano letivo 2021

Instruções

- É uma operação específica executada pelo processador
  - Transferência de dados

    ```
    MOV destino, fonte
    ```
  - Aritmética

    ```
    ADD destino, fonte
    ```
  - Lógica

    ```
    AND destino, fonte
    ```
  - Controle

    ```
    JMP local
    ```
  - Entre outras

Instruções

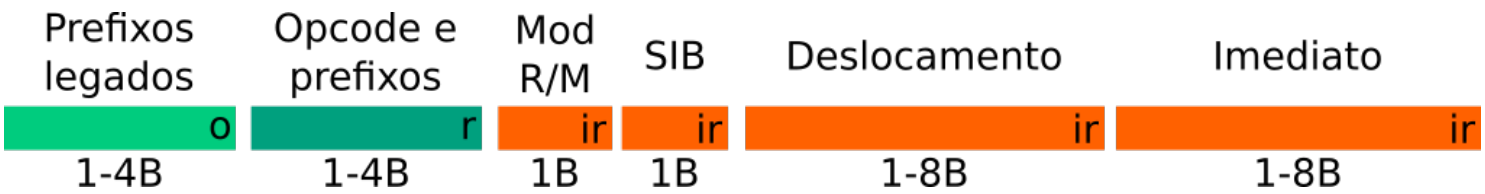
- Elementos de uma Instrução
  - Código da operação (*opcode*)

    Especifica a operação a ser executada
  - Referência à operando A (destino)

    Operando de retorno

    Também pode ser um operando fonte
  - Referência à operando B (fonte)

    Operando de entrada
- Instruções x86_x64 são complicadas
  - Pode conter até 15 bytes de extensão



Exemplos a08e01a/b.asm

- Código da Operação (*Opcode*)

a08e01a.asm:

```
6  section .text
7      global _start
8  _start:
9      mov eax, ebx
10 fim:
11     mov rax, 60
12     mov rdi, 0
13     syscall
```

a08e01b.asm:

```
6  section .text
7      global _start
8  _start:
9      add eax, ebx
10 fim:
11     mov rax, 60
12     mov rdi, 0
13     syscall
```

Exemplos a08e01a/b.asm

- Diferença entre os arquivos **.asm**:
  - Comando no terminal:

```
$ diff a08e01a.asm a08e01b.asm
```

  - Saída:

```
4c4
<     mov eax, ebx
---
>     add eax, ebx
```

- Diferença entre os arquivos montados/ligados **.x**:
  - Comando no terminal:

```
cmp -l a08e01a.x a08e01b.x
```

  - Saída:

```
129 211   1
720 141 142
d.  a.x b.x ; significado das colunas
```

Exemplos a08e01.a/b.asm

- A saída do comando *cmp* não é tão intuitiva, pois usa base decimal, por este motivo, alteramos a linha de comando anterior para:

```
cmp -l a08e01a.x a08e01b.x |
gawk '{printf "%08X %02X %02X\n",
$1-1, strtonum(0$2), strtonum(0$3)}'
```

  - obs.: necessário gawk. Quebra de linha é *evil*!

- Saída:

```
00000080   89   01
000002CF   61   62
    desl.  a.x  b.x ; significado das colunas
```

  - Para facilitar a vida dos alunos, existe o script *cmpASM.sh*, em anexo, que contém linha de comando.

```
chmod +x cmpASM.sh
```

  - 0x89 vs 0x01 $\rightarrow$ conferir *naquele* documento!

Exemplos a08e01.a/b.asm

- 00000080 89 01
  - Indica que no deslocamento 0x80, o arquivo a08e01a.x contém o valor 0x89, enquanto que o arquivo a08e01b.x contém o valor 0x01
    0x89 é parte do código da instrução MOV
    0x01 é parte do código da instrução ADD
- 000002CF 61 62
  - Indica que no deslocamento 0x2CF, o arquivo a08e01a.x contém o valor 0x61, enquanto que o arquivo a08e01b.x contém o valor 0x62
    0x61 e x062 representam os caracteres 'a' e 'b' na tabela ASCII

Exemplos a08e01.a/b.asm

- Editor Hexadecimal
  - É altamente recomendado procurar as diferenças entre os arquivos **.x** manualmente através do uso de um editor hexadecimal
    Editor Hexadecimal Online: ▸ HexEd.it
- Assembler/Disassembler Online *for fun*
  - ▸ defuse.ca

Instruções

- Referência à Operandos (MOD R/M)
  - Determina se os operandos da instrução são registradores (r), memória (m) ou imediato (i)
    Obs.: Definição extremamente simplificada
- As instruções, em sua maioria, utilizam 1 ou 2 operandos, que devem ser buscados em algum local de armazenamento ($r/m/i$), para então serem processados.
  - O local de armazenamento do operando é referenciado através de **Endereçamento**
  - Existem diversos **Modos de Endereçamento**
    - A adoção dos Modos de Endereçamento por uma arquitetura é decisão de projeto de instruções

Modos de Endereçamento

Importante:

- observaremos o operando `fonte`
- operando `destino` sempre será registrador `EAX`
- acompanhar com arquivo de exemplo `a08e02.asm`

Modos de Endereçamento:

- Imediato
- Direto
- Indireto
- por Registrador
- Indireto por Registrador
- por Deslocamento
- por Pilha (aula 09)

Endereçamento Imediato

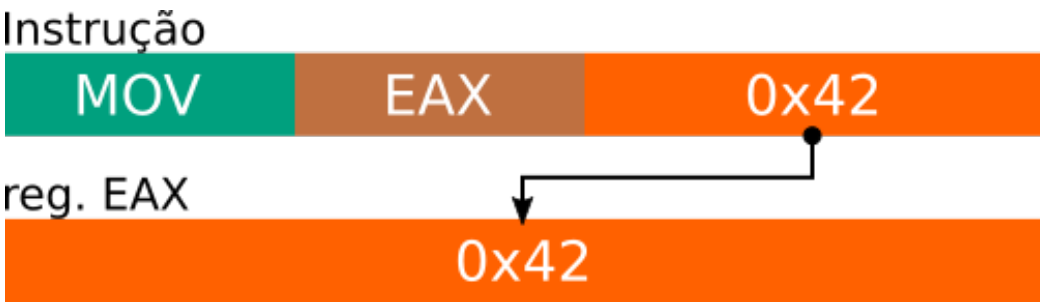
- Operando faz parte da instrução

Endereçamento imediato

  - Operando pronto no ciclo de busca à instrução
  - Usado na especificação de constantes
  - Forma mais simples de endereçamento
  - Não contém referência à memória
    - Operando limitado ao campo da instrução

Endereçamento Imediato - Exemplo

```
mov eax, 0x42
```



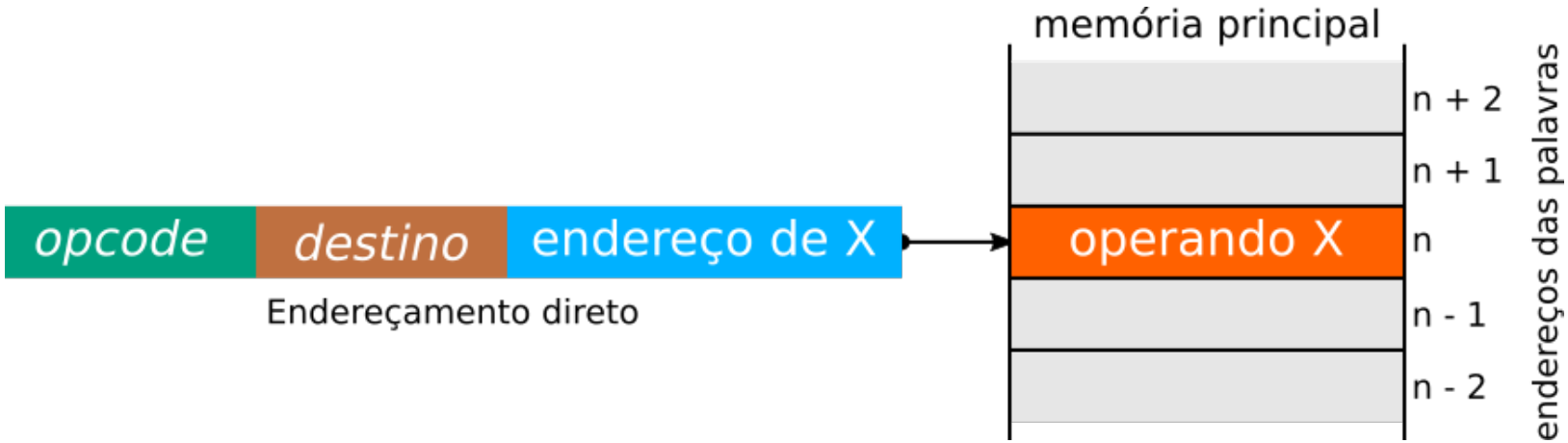
- gdb

```
b Imediato
r
x /xg Imediato
```

- é apresentado: 0x8bc03100000042b8

Endereçamento Direto

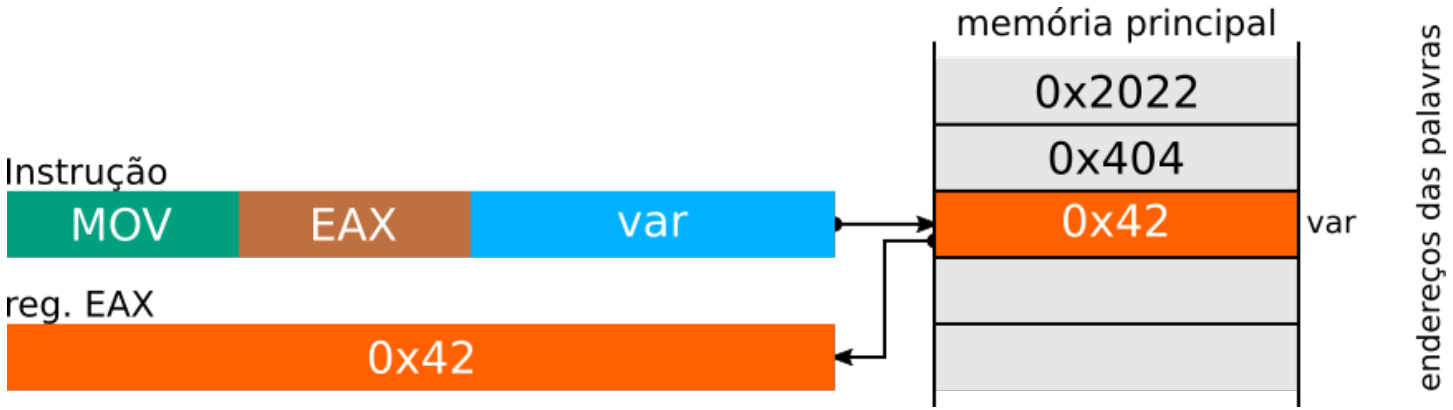
- Operando está na posição de memória indicado no campo da instrução



  - Contém 1 referência à memória
    - O intervalo referenciado é limitado pelo campo da instrução

Endereçamento Direto - Exemplo

```
mov eax, [var]
```



- gdb

```
b Direto
r
x /xg Direto
```

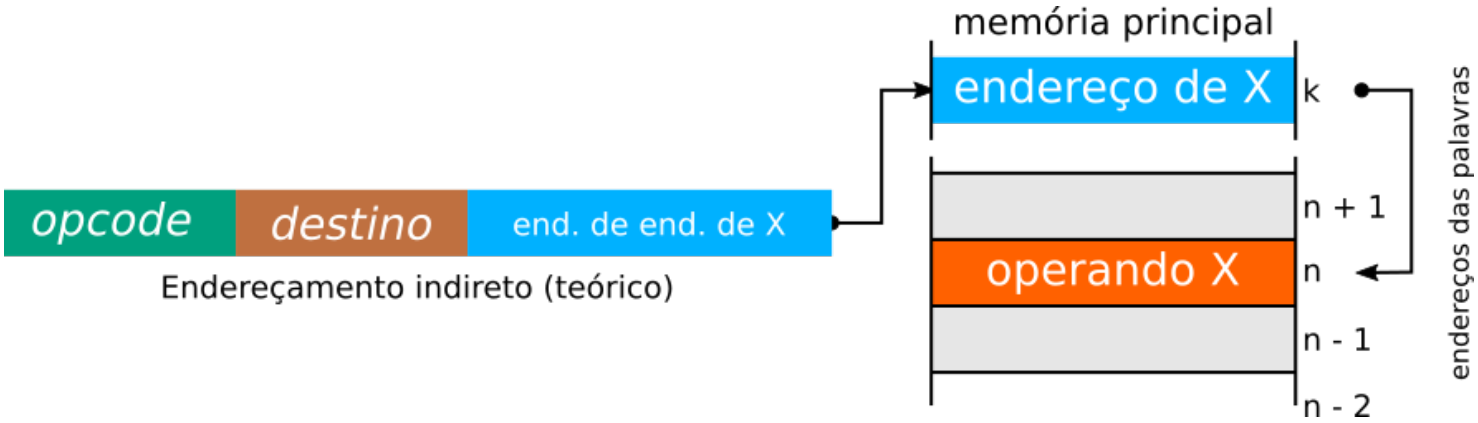
- é apresentado: 0x3100402000025048b

0x00402000 é endereço de `var`

```
x /xg &var
```

Endereçamento Indireto - conceito teórico

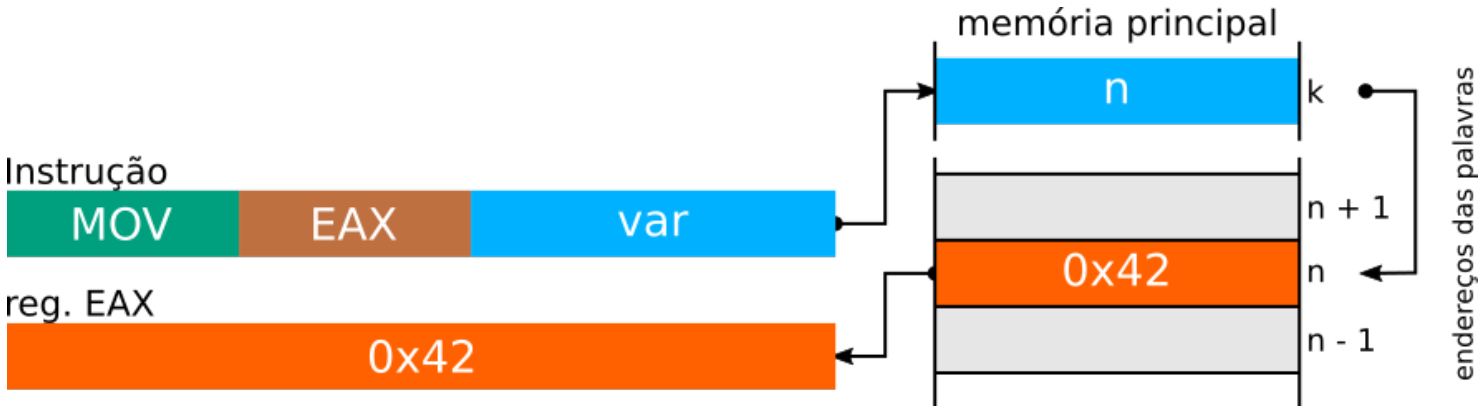
- Operando está na posição de memória indicado na posição de memória indicado no campo da instrução



  - Contém *n* referências à memória
    - O intervalo referenciado é limitado pelo campo da instrução na primeira indireção
    - O intervalo referenciado é limitado pela palavra de memória a partir da segunda indireção
  - **Não é suportado** pela arquitetura x86_64

Endereçamento Indireto - Exemplo teórico

```
mov eax, [[var]] ; novamente: nao suportado!
```



- simulação com ponteiro `n`:

```
mov qword [n], var
mov eax, [n]
```

Endereçamento por Registrador

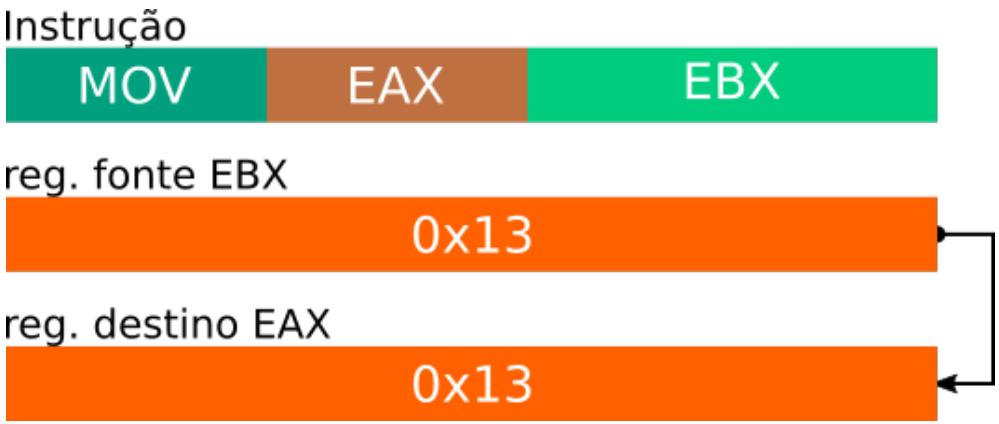
- Operando está no registrador indicado no campo da instrução

Endereçamento por registrador

  - Não contém referência à memória
    - Operando deve ter sido buscado/calculado previamente
    - Espaço de endereçamento é limitado ao número de registradores

Endereçamento por Registrador - Exemplo

```
mov eax, ebx
```

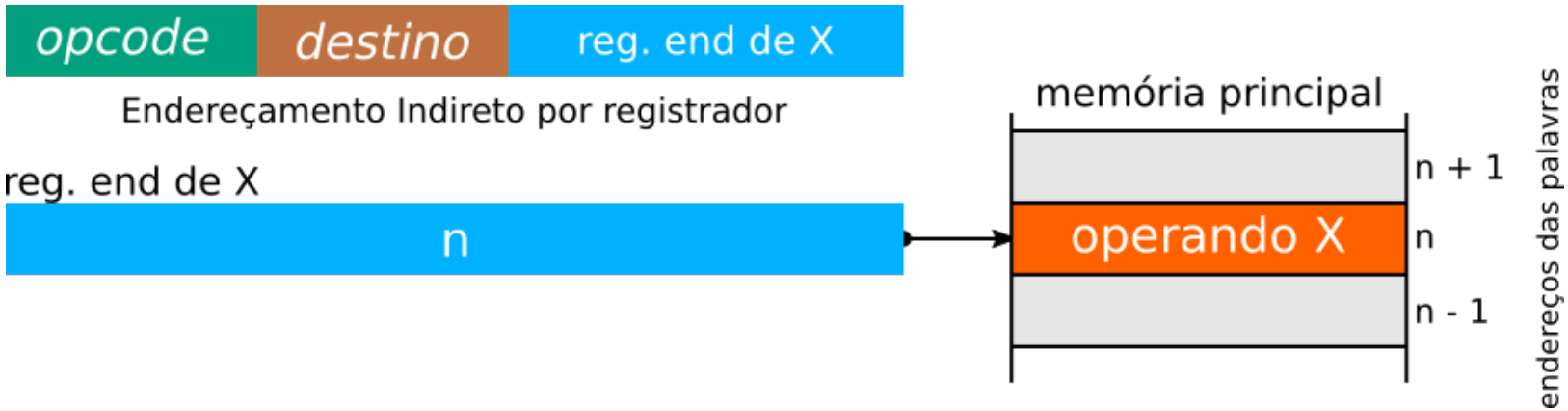


- gdb

```
b RegistradorP
r
p /x $eax
```

Endereçamento Indireto por Registrador

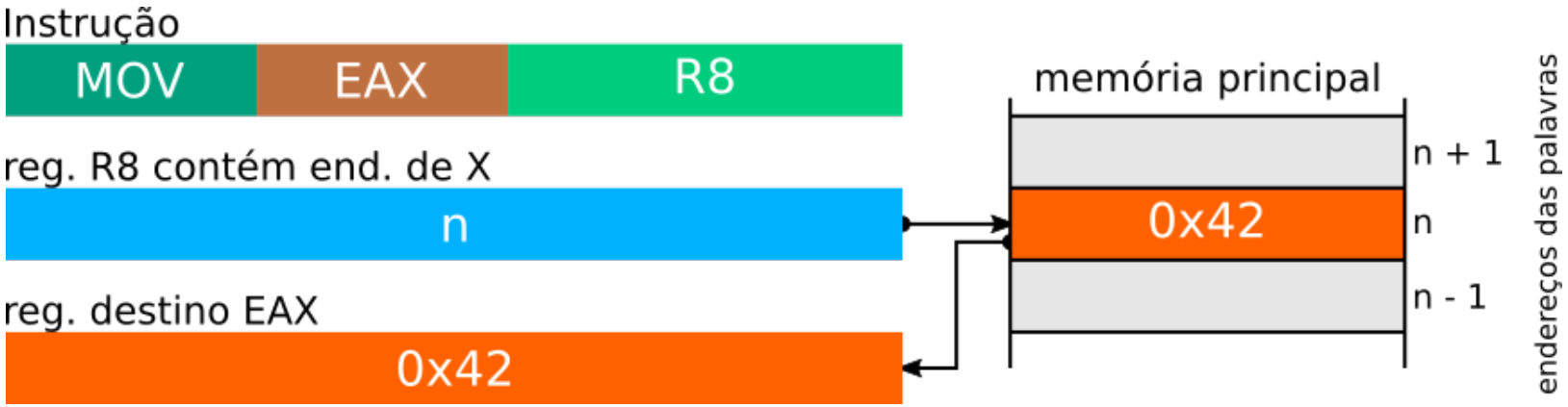
- Operando está na posição de memória indicado no registrador indicado no campo da instrução



  - Contém 1 referência à memória
    - Endereço deve ter sido buscado/calculado previamente

Endereçamento Indireto por Registrador - Exemplo

```
mov eax, [r8]
```

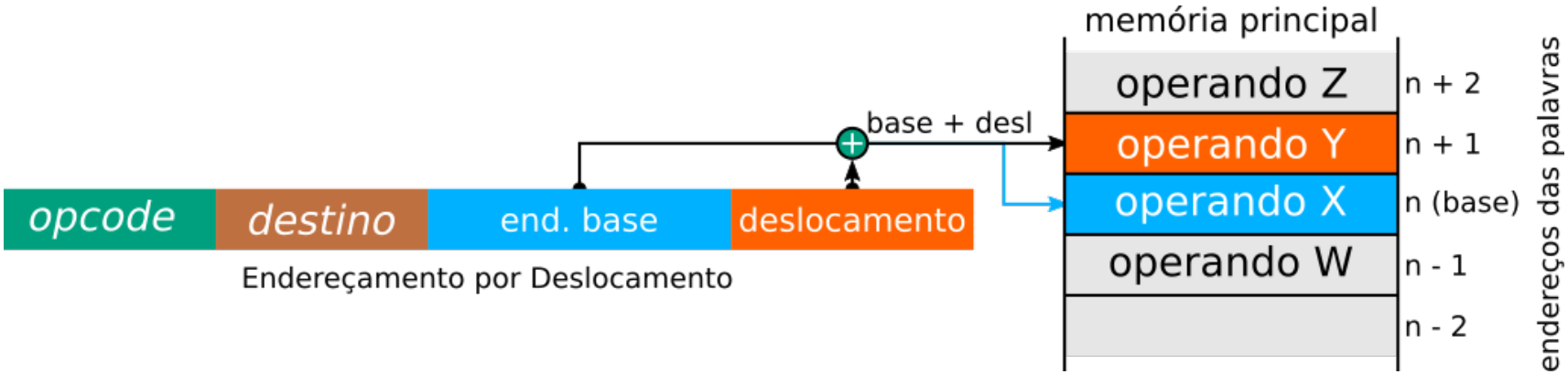


- gdb

```
b IndiretoRegistradorP
r
p /x $eax
```

Endereçamento por Deslocamento

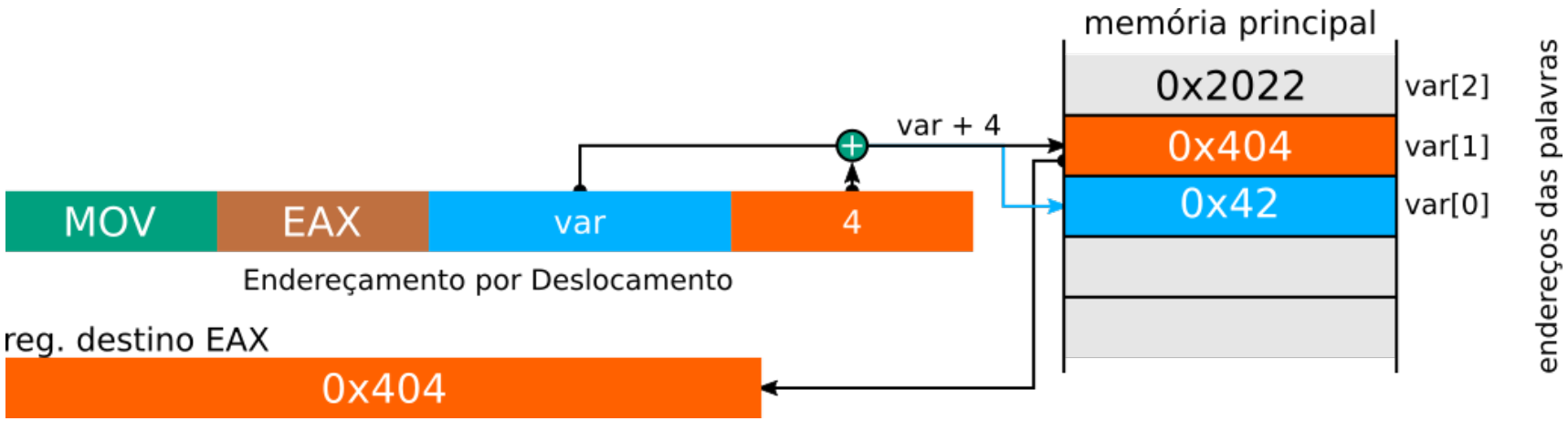
- Operando está na posição de memória resultante da adição entre o endereço base e um deslocamento definido na instrução
  - Deslocamento em Byte!



  - Contém 1 referência à memória
  - Requer cálculo do endereço do operando

Endereçamento por Deslocamento - Exemplo

```
mov eax, [var+4]
```



- gdb

```
b PorDeslocamento1
r
x /xg PorDeslocamento1
```

- é apresentado: 0x3100402004 25048b

  0x00402004 é endereço de `var[1]`

```
x /xw 0x402004
```

Endereçamento por Deslocamento (Variações)

Endereço Base + Deslocamento Imediato

```
        mov eax, [var+4] ; apresentado anteriormente
```

Endereço Base em Registrador + Deslocamento Imediato

```
        mov eax, [r8+4]
```

Endereço Base + Deslocamento em Registrador

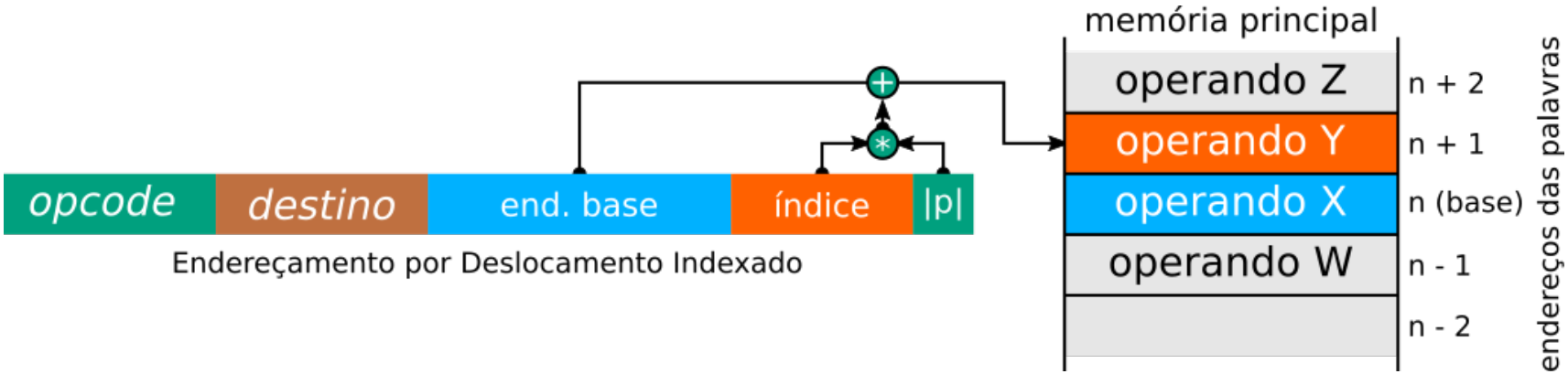
```
        mov eax, [var+ecx]
```

Endereço Base em Registrador + Deslocamento em Registrador

```
        mov eax, [r8+ecx]
```

Endereçamento por Deslocamento Indexado

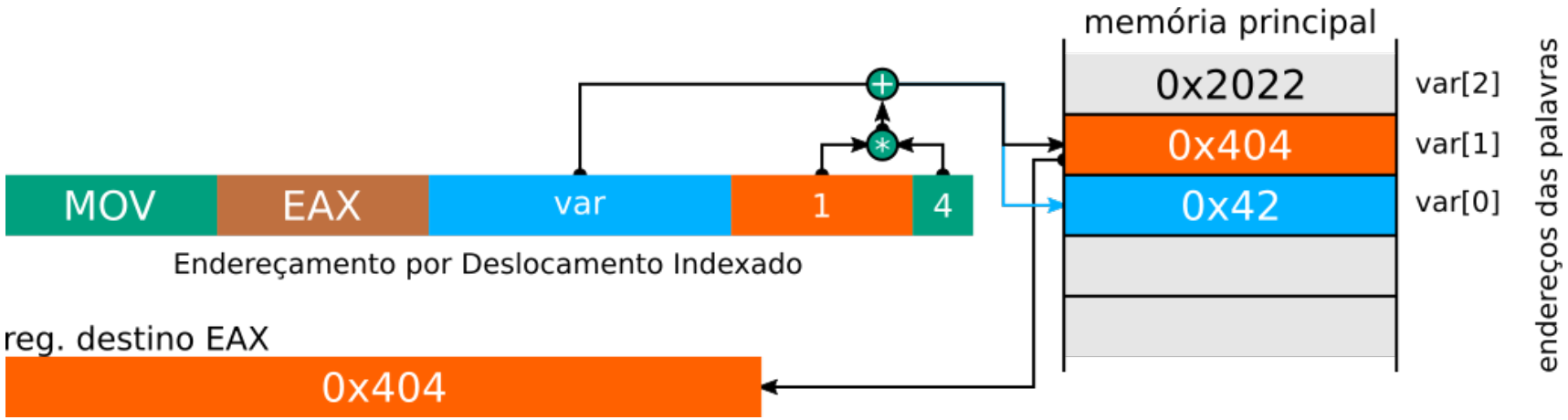
- Operando está na posição de memória resultante da adição entre o endereço base e um deslocamento indexado
- Deslocamento Indexado
  - cálculo entre o índice do deslocamento * tamanho da palavra |p|
  - é um deslocamento em `nível de palavra`



- Contém 1 referência à memória
- Requer cálculo do endereço do operando
- Utilizado para acesso à elementos de tipos de dados estruturados

Endereçamento por Deslocamento Indexado - Exemplo

```
mov eax, [var + 1 * 4]
```



- gdb

```
b PorDeslocamentoIndexado1
r
x /xg PorDeslocamentoIndexado1
```

- é apresentado: 0x3100402004025048b

0x00402004 é endereço de `var[1]`

```
x /xw 0x402004
```

Endereçamento por Deslocamento Indexado - Observações$_1$

PorDeslocamento vs PorDeslocamentoIndexado

- resultaram em 0x310040200425048b
- Qual é a diferença?
  - para exemplificar, foram usados constantes nos campos das instruções, que são detectadas pelo montador e calculadas antes de finalizar o arquivo objeto, por isso somos capazes de *debugar* esses valores iguais.

Endereçamento por Deslocamento Indexado - Observações$_2$

PorDeslocamento vs PorDeslocamentoIndexado

- desta forma, podemos inferir que ao usar registradores como deslocamento será gerado instruções diferentes
- novos exemplos:

```
lea r8, [var]
mov rcx, 4
mov eax, [r8+rcx]          ; 0x08048b41
```

```
lea r8, [var]
mov rcx, 1
mov eax, [r8 + rcx * 4] ; 0x88048b41
```

  - dica: use o defuse.ca

Endereçamento por Deslocamento Indexado - Observações$_3$

|p |

- disponível somente nos sabores 1, 2, 4 ou 8 (x86_64)
- formalmente chamado de *scale fator*
  - ao tentar usar valor diferente, o montador apresentará o seguinte erro:

```
error: invalid effective address
```

Quer saber mais sobre *Scale-Factor Index Base*?

- SIB em 2-22 até 2-24 Vol 1 (pdf págs 86 até 88) *daquele documento*
- ou aguarde a 3ª etapa da disciplina de OAC!

Endereçamento por Deslocamento Indexado (variações)

Endereço Base + Índice * Tamanho

```
        mov eax, [var + 1 * 4] ; apresentado anteriormente
```

Endereço Base + Índice em Registrador * Tamanho

```
        mov eax, [var + rcx * 4]
```

Endereço Base em Registrador + Índice * Tamanho

```
        mov eax, [r8 + 1 * 4]
```

Endereço Base em Registrador + Índice em Registrador * Tamanho

```
        mov eax, [r8 + rcx * 4]
```

Exemplo a08e03.asm

```
18      ...
19  laco:
20      mov al , [r8]       ; end. indireto por registrador
21
22      mov bl , [v1+r15d]     ; end. por deslocamento
23              ; offset + deslocamento (R)
24
25      ;mov ecx, [r9+r15d*4] ; end. eh invalido
26                            ; misturar 32b com 64b?
27
28      mov edx, [v2+r15d*4]  ; end. por desloc. indexado
29              ; offset + desloc. * posicionamento
30      ...
```

- GDB *time!*

Atividades Práticas

- a08at01: Utilize os exemplos a08e01a/b.asm para descobrir o código das seguintes instruções:
  - SUB r32/r32
  - OR r32/r32
  - AND r32/r32
  - XOR r32/r32
- AP0801-Extra: utilizando a instrução ADD, procure os valores dos registradores básicos EAX, EBX, ECX, EDX
  - Segundo byte da instrução ADD é MOD,REG,R/M (MOD R/M)
    - MOD: 2 bits e sempre será "11" para instrução ADD r32,r32
    - REG: 3 bits e é o registrador fonte
    - R/M: 3 bits e é o registrador destino

Atividades Práticas

- a08at02: Inversão de vetor de caracteres.
  - Dado um vetor de 10 posições de caracteres, inverta a ordem dos seus elementos
  - O vetor de caracteres deve ser lido do teclado
    - Considere que o usuário é esperto o suficiente para sempre entrar com 10 caracteres + < *enter* >
    - O < *enter* > não deve ter a posição alterada
    - ambos os vetores devem estar alocados ao mesmo tempo (use 2 variáveis não inicializadas)
  - Mostre a sequência de caracteres com a inversão
    - Exemplo de entrada/saída:

```
$: ./a08at02.x
entre com o vetor: abcdefghij<enter>
jihgfedcba eh inversao de abcdefghij
$:
```

Atividades Práticas

- a08at03: Ordenação de vetor de inteiros
  - Dado um vetor de 10 posições de inteiros (4 bytes cada), ordene seus elementos em ordem crescente
  - O vetor de inteiros deve ser criado como uma variável inicializada
    - Utilize 10 valores fora de ordem (inclusive iguais)
    - Pode ser utilizado qualquer algoritmo de ordenação desde que: A ordenação deve acontecer no mesmo vetor (e não em um vetor auxiliar)
    - O programa não tem iteração com o usuário, assim deve ser *debuggado* para confirmar a corretude.

Fim do Documento

Dúvidas?

Aula 09:

- Modo de Endereçamento em Pilha
- Passagem de Parâmetros via Linha de Comando
  - argc e argv