Programação Imperativa

Conceitos Elementares

Prof. Edson Alves

Faculdade UnB Gama

2020

Sumário

1. **Conceitos Elementares**

2. **Atribuição e Variáveis**

Programação Imperativa

- É uma abstração de computadores reais, os quais são baseados em máquinas de Turing e na arquitetura de Von Neumann, com registradores e memória
- O conceito fundamental é o de estados modificáveis
- Variáveis e atribuições são os construtos de programação análogos aos registradores dos ábacos e dos quadrados das máquinas de Turing
- Linguagens imperativas fornecem uma série de comandos que permitem a manipulação do estado da máquina

Estados

- **Nomes** podem ser associados a um valor e depois serem associados a um outro valor distinto
- O conjunto de nomes, de valores associados e a localização do ponto de controle do programa constituem o **estado** do programa
- O estado é um modelo lógico que associa localizações de memória à valores
- Um programa em execução gera uma sequência de estados
- As **transições** entre os estados é feita por meio de atribuições e comandos de sequenciamento
- A menos que sejam cuidadosamente escritos, programas imperativos só podem ser entendimentos em termos de seu comportamento de execução
- Isto porque, durante a execução, qualquer variável pode ser referenciada, o controle pode se mover para um ponto arbitrário e qualquer valor pode ser modificado

Valores, variáveis e nomes

- Os padrões binários que o hardware reconhece são considerados, nas linguagens de programação, **valores**
- Uma unidade de armazenamento em hardware equivale a uma **variável**, no ponto de vista do programa
- O endereço da unidade de armazenamento é interpretado como um **nome** no contexto de programação
- Assim, um nome é associado tanto ao endereço (localização) de uma unidade de armazenamento quanto ao valor armazenado nesta unidade
- A localização é denominada *l-value*, e o valor em si, *r-value*
- Por exemplo, na expressão:

$$x = x + 2;$$

o x à esquerda da expressão é um *l-value* (localização), enquanto o x da direita é um *r-value* (valor)

Atribuições

- Atribuições mudam os valores de uma dada localização
- Em Assembly, atribuições são feitas por meio do comando `MOV`:

```
MOV reg, reg
MOV reg, imm
```

- A sintaxe Assembly para comandos com dois parâmetros é a seguinte:

```
; Computa o número de vértices de um poliedro por meio da fórmula de Euler
;
;   V - E + F = 2
;
.SECTION text
global _start

_start:
    mov
```

onde `dest` é a localização onde será escrito o valor contido em `orig`

- Na segunda forma do comando `MOV`, `imm` refere-se a um valor **imediato**
- Esta valor corresponde a um número inteiro, em notação decimal ou hexadecimal (por meio do sufixo `H`)

Exemplo de atribuição

```
1  ; Exemplo de atribuição em Assembly
2
3  ; O resultado da execução deste programa pode ser visto no terminal
4  ; por meio do comando
5  ;
6  ;   $ echo $?
7  ;
8  SECTION .text
9  global  _start
10
11 _start:
12     mov ecx, 14H     ; O número 20 (em forma hexadecimal) para ECX
13     mov ebx, ecx     ; Copia o valor de ECX em EBX
14     mov eax, 1       ; Move o código de SYS_EXIT (opcode 1) para EAX
15     int 80h          ; Encerra o programa com erro (código 20)
```

Adição e subtração

- O valor a ser atribuído pode ser o resultado de uma das quatro operações aritméticas
- A **adição** e a **subtração** tem a mesma sintaxe:

```
ADD reg, reg
ADD reg, imm

SUB reg, reg
SUB reg, imm
```

- Na primeira forma, o valor armazenado em `orig` é adicionado/subtraído do valor contido em `dest`, e o resultado é armazenado em `dest`
- Na segunda forma, o valor do registrador é atualizado, através da adição/subtração do valor imediato
- Ao contrário da matemática, nenhum dos dois comandos é comutativo

Exemplo de aplicação da adição e da subtração

```
1  ; Computa o número de vértices de um poliedro por meio da
2  ; fórmula de Euler:
3  ;
4  ;   V - E + F = 2
5  ;
6  SECTION text
7  global _start
8
9  _start:
10     mov edx, 6       ; Número de arestas (E) em EDX
11     mov ecx, 4       ; Número de faces (F) em ECX
12
13     mov ebx, 2       ; O número de vértices (V) ficará armazenado em EBX
14     add ebx, edx
15     sub ebx, ecx
16
17     mov eax, 1       ; Move o código de SYS_EXIT (opcode 1) para EAX
18     int 80h          ; Encerra o programa com erro V = 4 (tetraedro)
```

Multiplicação

- A multiplicação não compartilha da mesma sintaxe da adição e da subtração
- Isto porque, ao multiplicar dois números de b-*bits*, o resultado será um número de $2b$-*bits*
- Assim, a sintaxe da multiplicação é

  `MUL reg`
- Se reg é um registrador de 8-*bits*, este será multiplicado por `AL` e o produto será armazenado em `AX`
- Por exemplo,

  `MUL BH`

  equivale a `AX` = `AL`·`bh`

Referências

1. asmtutor.com. Learn Assembly Language, acesso em 16/01/2020.
2. NASM. Site Oficial, acesso em 16/01/2020.
3. **NEVELN**, Bob. *Linux Assembly Language Programming*, Open Source Technology Series, Prentice-Hall, 2000.
4. **SHALOM**, Elad. *A Review of Programming Paradigms Througout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.
5. The Geometry Junkyard. Twenty Proofs of Euler's Formula: V - E + F = 2, acesso em 21/01/2020.