Programação Imperativa

Condicionais

Prof. Edson Alves

Faculdade UnB Gama

2020

Sumário

Saltos

- As atribuições simulam as instruções de escrever ou apagar um traço nas fitas das máquinas de Turing
- Assim, as linguagens imperativas precisam também de mecanismos que permitam que o controle decida o próximo estado a ser avaliado de forma condicional, de acordo com o estado atual
- Nas linguagens Assembly isto é feito por meio de saltos condicionais
- Os saltos são instruções que desviam o controle para pontos específicos, identificados por rótulos, de acordo com o estado do registrador de controle
- Como podem existir diversas combinações das variáveis (*flags*) que são consideradas no registrador de controle, há várias instruções de salto distintas

Salto incondicional

- A instrução `JMP` corresponde a um salto incondicional
- A sintaxe desta instrução é

```
JMP label
```

- *label* corresponde a um rótulo, e a execução do programa seguirá para a primeira instrução que segue o rótulo
- Como o salto é incondicional, a depender do posicionamento da instrução de salto e do rótulo o programa pode ficar preso em um laço infinito, jamais encerrando sua execução
- Ainda assim, saltos incondicionais são úteis, principalmente para sair de laços aninhados ou para encerrar o programa a partir de qualquer ponto

Saltos condicionais

- Um salto condicional avalia uma ou mais *flags* e, a depender dos estados delas (0 ou 1), realiza o salto ou não
- A instrução `JZ` salta para *label* se a *flag* zero for igual a 1
- As instruções `ADD` e `SUB` modificam esta *flag*, tornando igual a 1 se o resultado da operação é igual a zero, ou 0, caso contrário
- A instrução `JNZ` salta para *label* se a *flag* zero for igual a 0
- Outra *flag* que é modificada pelas instruções `ADD` e `SUB` é a de sinal, que se torna um se o resultado da opereação é negativo, ou zero, caso contrário
- As instruções `JS` e `JNS` são semelhantes às instruções `JZ` e `JNZ`, porém elas avaliam a *flag* de sinal

Exemplo de saltos condicionais

```
1  ; Computa o número de raizes reais do polinômio
2  ;
3  ;   p(x) = ax^2 + bx + c
4  ;
5  ; Como exemplo, serão utilizados os valores a = 1, b = -5 e c = 6
6  SECTION .data
7  a       dd  1               ; dd = variáveis com 4 bytes
8  b       dd  2               ; (use dw para 2 bytes)
9  c       dd  3
10 msg     db  '0', 0Ah, 0
11
12 SECTION .text
13 global _start
14
15 _start:
16     mov eax, 4       ; EBX = 4*a*c
17     mov ecx, [a]
18     mul ecx
19     mov ecx, [c]
20     mul ecx
21     mov ebx, eax
22
```

Exemplo de saltos condicionais

```
23      mov eax, [b]     ; b = -5
24      mul eax          ; EAX = b^2
25
26      sub eax, ebx     ; EAX = b^2 - 4*a*c
27      jz  one          ; Se EAX = 0 há apenas uma raiz
28
29      sub eax, 0
30      jns two          ; Se EAX > 0 há duas raizes reais
31
32      mov ebx, '0'     ; Caso contrário não há raizes reais
33      jmp finish
34
35  one:
36      mov ebx, '1'
37      jmp finish
38
39  two:
40      mov ebx, '2'
41
42  finish:
43      mov [msg], bl    ; Atualiza a mensagem com o número de raizes
```

Exemplo de saltos condicionais

```
44
45      mov edx, 3          ; msg tem 3 bytes
46      mov ecx, msg        ; msg é a mensagem a ser impressa
47      mov ebx, 1          ; A saída é STDOUT
48      mov eax, 4          ; Optcode de sys_write
49      int 80h
50
51      mov ebx, 0          ; Encerra com sucesso
52      mov eax, 1
53      int 80h
```

Comparações

- Outra *flag* que é modificada pelas operações aritméticas é a de *overflow*
- As diferentes combinações das *flags* de sinal, de *overflow* e zero permite simular os operadores relacionais das linguagens de programação de alto nível
- A instrução `CMP`, cuja sintaxe é

  ```
  CMP x, y
  ```

  compara o conteúdo dos registradores x e y, modificando as *flags* apropriadas
- Após esta instrução, é possível utilizar um dos comandos de saltos listados na tabela a seguir, se x e y forem inteiros sinalizados (para não sinalizados há outro conjunto de saltos)
- Veja que, se após a instrução `CMP` e antes do salto, for executada alguma instrução que modifique alguma das *flags*, o salto pode não ter o efeito esperado

Saltos associados aos operadores relacionais

Instrução	Efeito
JL	Salta se $x < y$
JLE	Salta se $x \leq y$
JG	Salta se $x > y$
JGE	Salta se $x \geq y$
JE	Salta se $x = y$
JNL	Salta se $x \not< y$
JNLE	Salta se $x \not\leq y$
JNG	Salta se $x \not> y$
JNGE	Salta se $x \not\geq y$
JNE	Salta se $x \neq y$

Exemplo de comparações

```
1  ; Determina se o número n é par ou ímpar
2  SECTION .data
3  n       dd  5
4  even    db  'Par', 0Ah, 0
5  odd     db  'Impar', 0Ah, 0
6
7  SECTION .text
8  global _start
9
10 _start:
11     mov eax, [n]     ; a = n
12     mov ebx, 2       ; b = 2
13     div ebx          ; r = edx, q = eax
14
15     cmp edx, 0       ; Testa se n é par (r == 0)
16     je  par
17
18     mov edx, 7       ; n é ímpar
19     mov ecx, odd
20     jmp finish
21
```

Exemplo de comparações

```
22 par:
23     mov edx, 5       ; even tem 5 bytes
24     mov ecx, even
25     jmp finish
26
27 finish:
28     mov ebx, 1       ; A saída é STDOUT
29     mov eax, 4       ; Optcode de sys_write
30     int 80h
31
32     mov ebx, esi       ; Encerra com sucesso
33     mov eax, 1
34     int 80h
```

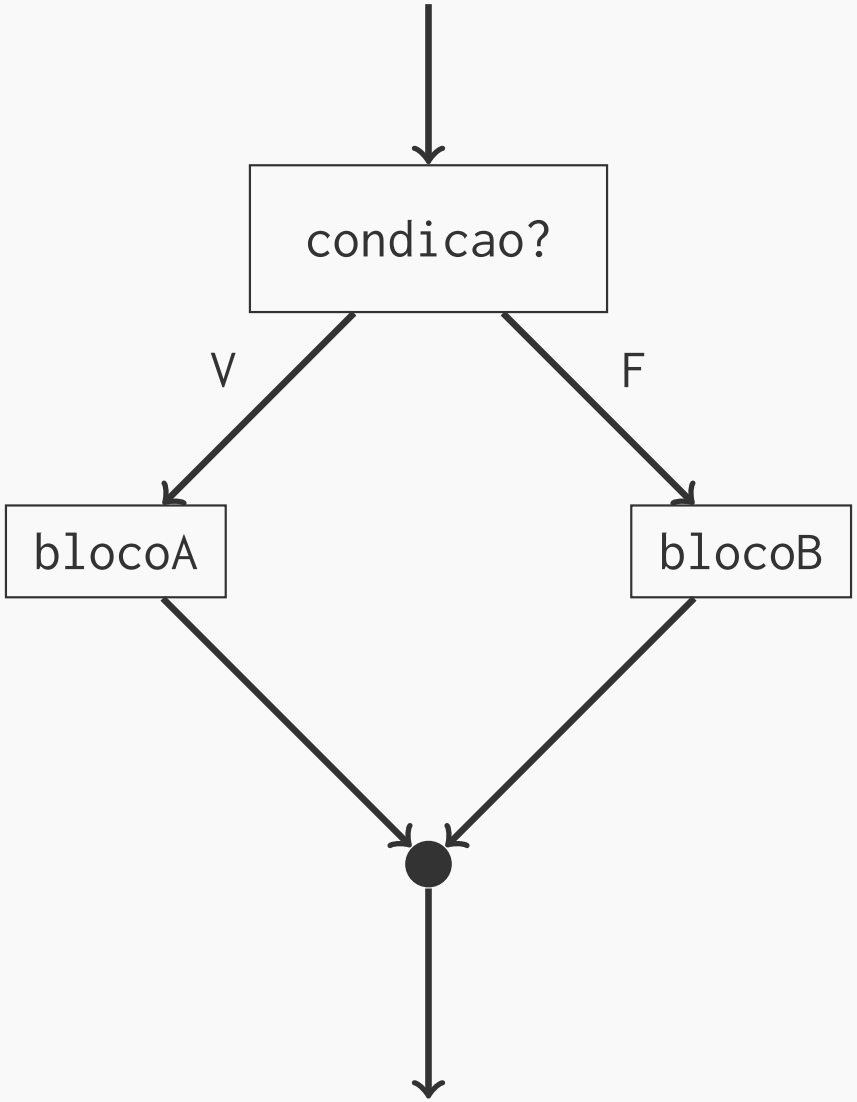
IF-ELSE

- Os saltos condicionais permitem simular estruturas de controle presentes em outras linguagens
- O construto IF-ELSE tem a seguinte sintaxe

```
if condicao then
    blocoA
else
    blocoB
```

- Se a condicao for avaliada como verdadeira, são executados os comandos associados ao blocoA; caso contrário, são executados os comandos do blocoB
- A cláusula ELSE é opcional
- Este construto desvia a execução do programa, que a partir da condição passa a ter dois caminhos possíveis, e estes caminhos são mutuamente exclusivos
- Após o término do bloco escolhido, a execução continua do ponto que segue o último comando associado a blocoB

Visualização do construto IF-ELSE



Codificação do construto IF-ELSE em Assembly

```
1  ; Código correspondente ao construto IF-ELSE
2
3      cmp regA, regB      ; Esta comparação corresponde à condição
4      jnz blockA          ; Assuma 0 falso, caso contrário verdadeiro
5      jmp blockB
6
7  blockA:
8      ; Commandos associados ao blocoA
9      jmp finish
10
11 blockB:
12     ; Commandos associados ao blocoB
13     jmp finish
14
15 finish:
16     ; Prossegue com a execução do código
```

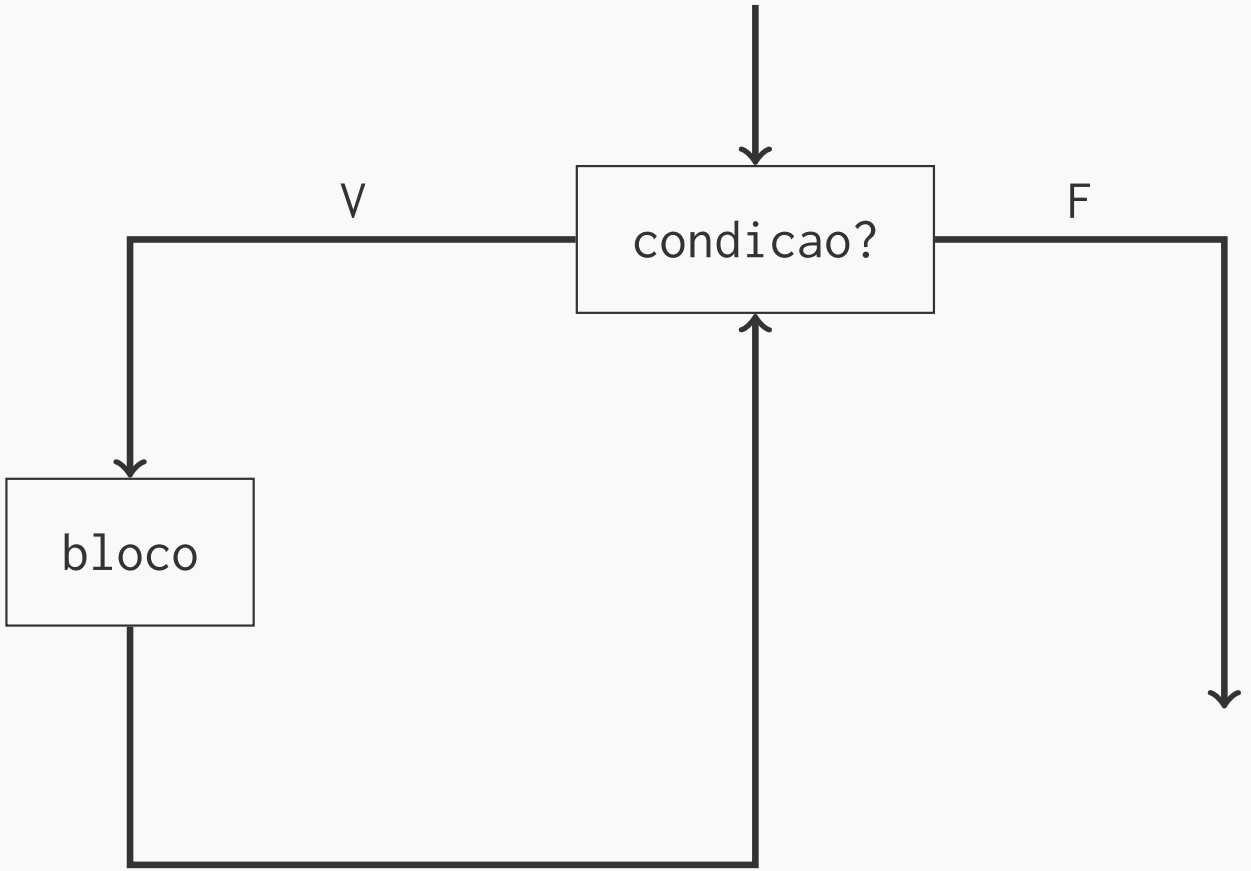
WHILE

- Outro construto que pode ser simulado com saltos condicionais é o laço WHILE
- A sintaxe do laço WHILE é

```
while condicao do
    bloco
```

- Os comandos associados ao bloco serão executados sempre que a condicao for verdadeira
- Deste modo, caso os comandos do bloco não alterem o estado de modo a permitir que a condição se torne falsa, o laço se repetirá indefinidamente
- A presença de saltos dentre os comandos do bloco permite a saída prematura do bloco

Visualização do construto WHILE



Exemplo de laços e condicionais

```
1  ; Determina se o número n é primo ou não
2  SECTION .data
3  yes     db  ' eh primo', 0Ah, 0
4  no      db  ' nao eh primo', 0Ah, 0
5
6  SECTION .bss
7  bf:     resb 256    ; Buffer de leitura
8
9  SECTION .text
10 global _start
11
12 _start:
13     ; Lê um número do console
14     mov edx, 256    ; Lê, no máximo, 256 caracteres
15     mov ecx, bf     ; Grava a leitura em bf
16     mov ebx, 0      ; Lê de STDIN
17     mov eax, 3      ; Optcode de sys_read
18     int 80h         ; EAX = digitos lidos + '\n'
19
20     dec eax         ; Desconta o '\n'
21
```

Exemplo de laços e condicionais

```
22      ; Imprime o número lido
23      mov edx, eax    ; Número de caracteres lidos
24      mov ecx, bf     ; Buffer de leitura
25      mov ebx, 1      ; Escreve em STDOUT
26      mov eax, 4      ; Optcode de sys_write
27      int 80h
28
29      ; Converte a string para inteiro
30      mov eax, 0      ; EAX conterá o número convertido
31      mov ebx, 10     ; Base numérica
32      mov ecx, bf     ; Buffer contendo o número como string
33
34  to_int:
35      mov edx, 0      ; Extrai o próximo dígito
36      mov dl, [ecx]
37
38      cmp dl, '0'     ; Se o caractere está fora da faixa [0-9] finaliza
39      jl done
40
41      cmp dl, '9'
42      jg done
43
```

Exemplo de laços e condicionais

```
44      sub edx, '0'     ; Converte de ASCII para decimal
45      mul ebx          ; EAX = 10*EAX + EDX
46      add eax, edx
47
48      inc ecx          ; Avança o ponteiro e continua o laço
49      jmp to_int
50
51  done:
52      mov esi, eax     ; ESI = n
53
54      ; Verifica se o número é menor que 2
55      mov ebx, 2
56      cmp esi, ebx
57      jl not_prime
58
59      ; Verifica se é igual a 2
60      je is_prime
61
62      ; Verifica se é par
63      mov eax, esi
64      div ebx
65      cmp edx, 0
66      je not_prime
```

Exemplo de laços e condicionais

```
67
68      ; Tenta todos os ímpares menores que a raiz quadrada
69      mov ecx, 3
70
71  next:
72      ; Checa se há ultrapassou a raiz quadrada
73      mov eax, ecx
74      mul eax
75      cmp eax, esi
76      jg is_prime
77
78      ; Verifica se ECX divide n
79      mov eax, esi
80      div ecx
81      cmp edx, 0
82      je not_prime
83
84      ; Tenta o próximo ímpar
85      add ecx, 2
86      jmp next
87
```

Exemplo de laços e condicionais

```
88 is_prime:
89     mov edx, 11      ; 'yes' tem 11 bytes
90     mov ecx, yes     ; Escreve o conteúdo de no
91     jmp finish
92
93 not_prime:
94     mov edx, 15      ; 'no' tem 15 bytes
95     mov ecx, no      ; Escreve o conteúdo de no
96     jmp finish
97
98 finish:
99     mov ebx, 1       ; Escreve em STDOUT
100    mov eax, 4       ; Optcode de sys_write
101    int 80h
102
103    mov ebx, 0       ; Encerra com sucesso
104    mov eax, 1
105    int 80h
```

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.