Estruturas de Seleção

João Marcelo Uchôa de Alencar

6 de abril de 2023

1/44

Introdução

Estrutura *If-Then*

Estrutura *If-Then-Else*

Ifs Aninhados

Estrutura *Case*

Operações Lógicas e Caracteres

Expressões Aritméticas em Diretivas

Programa Completo

Resumo

2/44

Introdução

- Dois tipos principais de estruturas de controle:
  - Estruturas de seleção (*ifs*).
  - Estruturas de iteração (laços).
- No nível menos abstrato, todas as essas estruturas podem ser construídas com *ifs* e desvios (saltos):
  - Condicionais: desvio para uma instrução se determinada condição for verdadeira.
  - Não condicionais: desvio para uma instrução independente de qualquer condição.
- Em linguagens de alto nível, o *goto* é evitado. Em *assembly*, ele é indispensável.

3/44

Introdução

- MASM permite utilizar estruturas de controle parecidas com as de linguagens de alto nível, com palavras reservadas com *if* e *then*.
- Vamos começar por essas estruturas, depois mostramos como podem ser construídas em um nível mais básico.
- Apesar de facilitar o aprendizado, as estruturas de mais alto nível tem limitações:
  - Números de ponto flutuante não podem ser utilizados para controlar a execução.
  - Não podem ser usadas no modo 64 *bits*.

4/44

Estrutura *If-Then*

Em C:

```
if (number == 0)
   number--;
```

Em *assembly*:

```
.if number == 0
dec number
.endif
```

```
if (amount  != 1) {
   count++;
   amount = amount + 2;
}
```

```
.if amount != 1
inc count
add amount,2
.endif
```


5/44

Estrutura *If-Then*

- O ponto decimal antes de *if* e *endif* serve para indicar que são diretivas, não instruções.
- Os parênteses são opcionais nas condições.
- As condições relacionais disponíveis em C estão disponíveis em MASM.
- O uso de *.endif* é obrigatório.
- Usando registradores:
  - Um registrador pode ser comparado com um literal.
  - Dois registradores podem ser comparados entre si.
  - MASM assume que os registradores contém valores **sem sinal**.
- Para fazer comparações com números negativos, usamos **variáveis**.

6/44

Estrutura *If-Then*

Considere em C:

```
if (count > number)
    flag = -1;
```

- Duas posições de memória não podem ser comparadas em *.if*.
- A razão é que a instrução *cmp* não pode referenciar duas posições de memória.

Em *assembly*:

```
mov eax, count
.if eax > number
mov flag, -1
.endif
```


7/44

Estrutura *If-Then*

- O registrador *eflags* controla vários aspectos da CPU e contém o *status* da CPU.
- Não é possível acessar *eflags* diretamente.
- Várias instruções configuram *flags* de 1 ou 2 *bits*.
- Cada *flag* recebe uma abreviação individual de duas letras.
- Vamos usar operadores especiais para recuperar o *status* das *flags*.

8/44

Estrutura *If-Then*

Flag	Abreviação	Operador	Posição do *Bit*	Significado quando 1
Carry	CF	CARRY?	0	*bit carry on* de inteiro sem sinal
Parity	PF	PARITY?	2	Número par de conjunto de *bits*
Zero	ZF	ZERO?	6	O resultado a operação é zero
Sign	SF	SIGN?	7	O resultado é negativo
Direction	DF		10	Processar *strings* de cima para baixo
Overflow	OF	OVERFLOW?	11	*Overflow* de inteiro com sinal

De acordo com o resultado da última instrução:

- Se o resultado for zero, ZF tem valor 1 e SF tem valor 0.
- Se o resultado for negativo, SF tem valor 1 e ZF tem valor 0.
- Se o resultado for positivo, ZF e SF são 0.

9/44

Estrutura *If-Then*

- A instrução *cmp* compara dois operandos e atualiza *eflags*.
- Os dois operandos são subtraídos, mas os valores originais são mantidos.
- Por exemplo, se os dois operandos tem o mesmo valor, o resultado é zero.
- Se o primeiro operando for maior que o segundo, o resultado seria positivo.

10/44

Estrutura *If-Then*

Instrução	Significado
cmp reg,imm	Compara registrador e valor imediato
cmp imm,reg	Compara imediato e registrador
cmp reg,mem	Compara registrador e valor imediato
cmp mem,reg	Compara memória e registrador
cmp mem,imm	Compara memória e valor imediato
cmp imm,mem	Compara imediato e memória
cmp reg,reg	Compara registrador e registrador

Uma vez dois que dois operandos forem comparados, as *flags* correspondentes serão configuradas e desvios podem ser feitos de acordo com os valores delas.

11/44

Estrutura *If-Then*

je e *jne* podem ser usadas com inteiros com ou sem sinal:

Instrução	Significado
je	Desvia se igual
jne	Desvia se não igual

Instruções para dados com sinal:

Instrução	Significado	Instrução	Significado
jg	Desvia se maior	jnle	Desvia se não for menor ou igual
jge	Desvia se maior ou igual	jnl	Desvia se não for menor
jl	Desvia se menor	jnge	Desvia se não for maior ou igual
jle	Desvia se menor que ou igual	jng	Desvia se não for maior

As instruções da direita são equivalente às da esquerda, portanto não são tão usadas.

12/44

Estrutura *If-Then*

Voltando ao exemplo com as diretivas:

```
.if number == 0
dec number
.endif
```

A versão usando apenas instruções:

```
          ; if number == 0
if01:     cmp number,0 ; compare number and zero
          jne endif01  ; jump not equal to endif01
then01:   dec number   ; decrement number by one
endif01: nop           ; end if, no operation
```


13/44

Estrutura *If-Then*

- Na versão com diretivas, se $number == 0$, a próxima instrução após o *.if* é executada.
- Se $number! = 0$, a instrução após *.endif* é executada.
- Infelizmente, as instruções de desvio saltam quando o resultado é verdadeiro e não executam a instrução logo em seguida, mas sim a instrução indicada pelo rótulo passado como operando.
- A solução mais fácil é inverter a condição.

14/44

Estrutura *If-Then*

Voltando a um exemplo com as diretivas:

```
mov eax,count
.if eax > number
mov flag,-1
.endif
```

Usando instruções:

```
; if count > number
if02:     mov eax,count
          cmp eax, number
          jle endif02
then02:   mov flag,-1
endif02: nop
```

O oposto de **maior que** não é **menor que**, mas sim **menor ou igual que**.

15/44

Estrutura *If-Then-Else*

Em C:

```
if x >= y
   x--;
else
   y--;
```

Em *assembly*, com diretivas:

```
; if x >= y
mov eax,x
.if eax >= y
dec x
.else
dec y
endif
```


16/44

Estrutura *If-Then-Else*

```
; if x >= y
if03:    mov eax,x
         cmp eax,y
         jl else03
then03:  dec x
         jmp endif03
else03:  dec y
endif03: nop
```

Não podemos esquecer o *jmp* ao final da seção *then*, caso contrário a seção *else* será sempre executada.

17/44

Ifs Aninhados

Em C:

```
if (x < 50)
   y++;
else
   if (x <= 100)
      y = 0;
   else
      y--;
```

Em *assembly*:

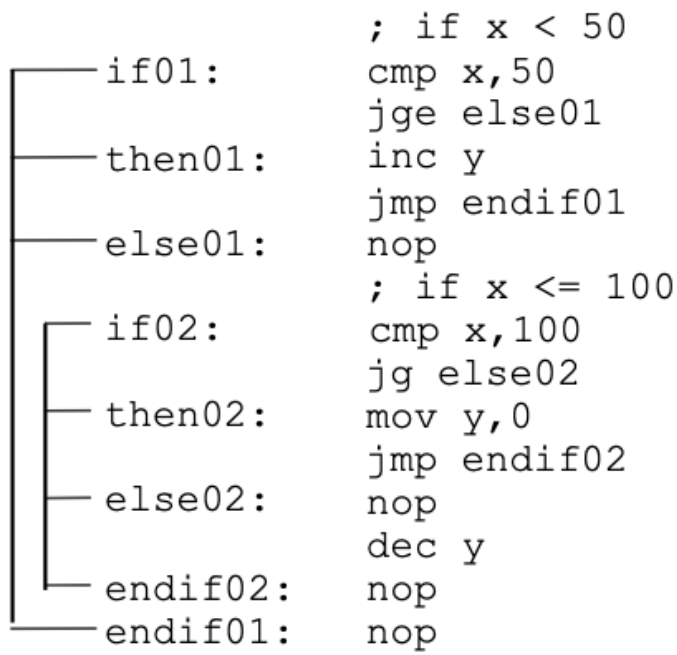
```
.if x < 50
inc y
.else
.if x <= 100
mov y,0
.else
dec y
.endif
.endif
```


18/44

Ifs Aninhados

- Existem rótulos próprios para cada *if-then-else*.
- Dois *.endif* são necessários.
- Poderíamos desviar diretamente para *if02*, mas *jge else01* ajuda na legibilidade.
- As instruções *nop* não são obrigatórias, mas poderiam ser substituídos por outras instruções.

Versão com a instrução *cmp*:



É sempre importante lembrar que os rótulos são necessários, mas no código binário final não estão presentes. São substituídos por endereços. Portanto, não ocupam memória.

19/44

Diretiva *.elseif*

Podemos usar o *.elseif* para eliminar um dos *.endif*:

```
.if (x < 50)
inc y
.elseif (x <= 100)
mov y,0
.else
dec y
.endif
```

Caso você precise adicionar código antes ou depois do *if* interno, *.elseif* deve ser evitado.

20/44

Estrutura *if-then-if*

Em C:

```
if (x <= 100)
    if (x < 50)
        y++;
    else
        y = 0;
else
    y--;
```

Em *assembly*:

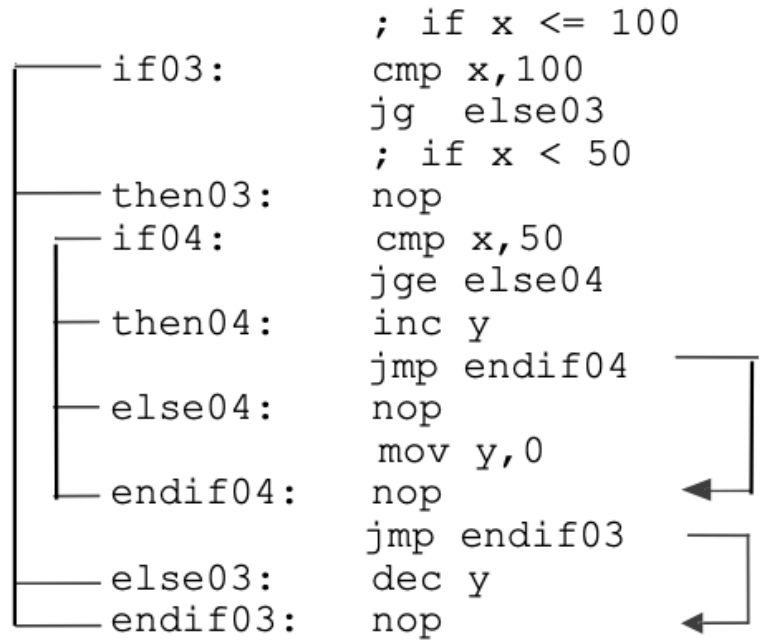
```
.if x <= 100
.if x < 50
inc y
.else
mov y,0
.endif
.else
dec y
.endif
```


21/44

Estrutura *if-then-if*

- Sempre lembrar que a condição nos desvios é inversa ao código em C.
- Os desvios incondicionais também são importantes.
- Consegue observar que o código tem dois desvios em sequência?

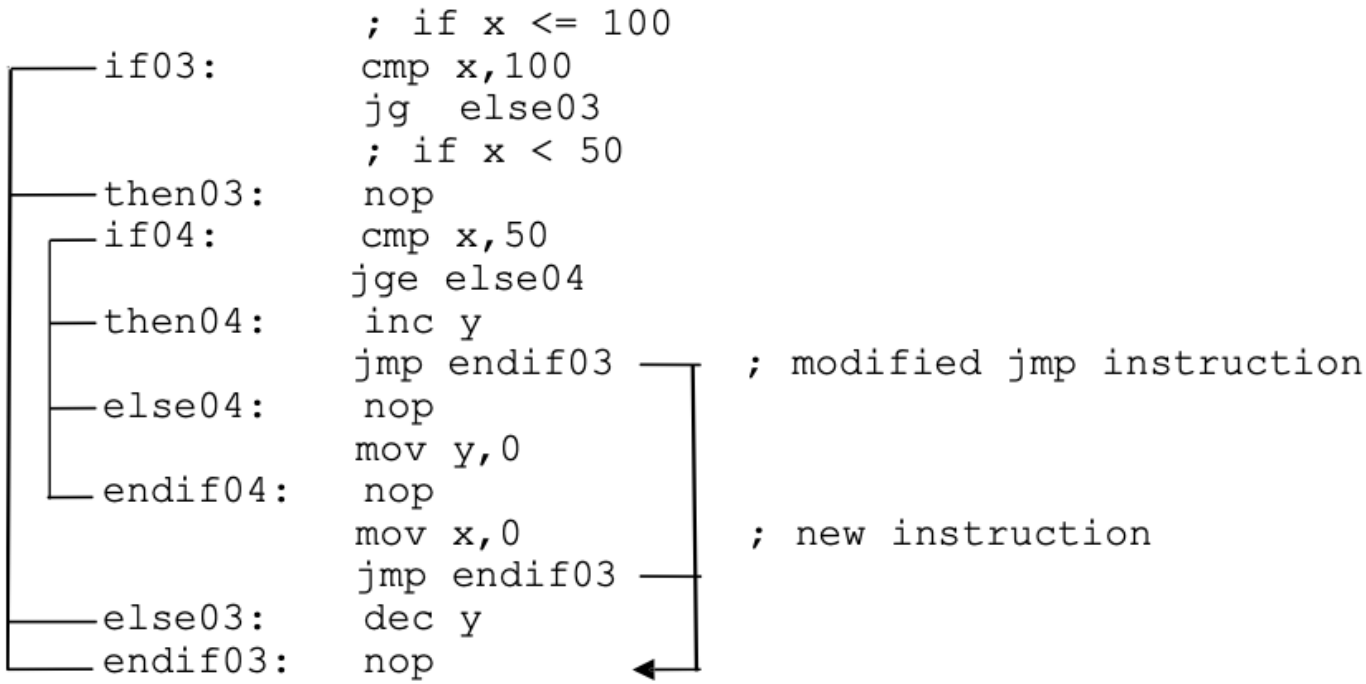
Versão com a instrução *cmp*:



Suponha que, para todos os casos que $x <= 100$, quiséssemos executar $x = 0$ após a verificação $x < 50$. Bastaria colocar *mov x, 0* após *nop* no rótulo *endif04*.

22/44

Estrutura *if-then-if*



É melhor manter alguns desvios que podem parecer desnecessários para manter a legibilidade.

23/44

Estrutura *Case*

Não existe *switch* em MASM. Considere o fragmento em C:

```
switch (w) {
        case 1: x++;
                break;
        case 2:
        case 3: y++;
                break;
        default: z++;
}
```

Para implementá-lo em MASM, precisamos utilizar uma série de instruções *cmp* e *je*.

24/44

Estrutura *Case*

```
switch01:     cmp w,1
              je case11;
              cmp w,2
              je case12;
              cmp w,3
              je case12;
              jmp default01
case11:       inc x
              jmp endswitch01
case12:       inc y
              jmp endswitch01
default01:    inc z
endswitch01: nop
```


25/44

Operações Lógicas e Caracteres

Em C, assumindo que *initial* é do tipo *char*:

```
if (initial < 'e')
    count++
```

Em *assembly*, considerando *initial* como *byte*:

```
.if initial < 'e'
inc count
.endif
```

Precisamos cuidar de alguns detalhes para implementação com *cmp*.

26/44

Operações Lógicas e Caracteres

- As instruções *jg*, *jge*, etc são para números com sinal.
- O binário de 'a' é 01000001, o binário de 'b' é 01000010. Como o primeiro *bit* é 0, não teria problema em comparar usando as instruções já vistas.
- Agora considerando ASCII estendido, *ä* tem como binário 10000100. Seria um número negativo.
- Existem instruções específicas para saltos após comparações com valores sem sinal.

27/44

Operações Lógicas e Caracteres

Instrução	Significado	Instrução	Significado
ja	Desvia acima	jnbe	Desvia se não for abaixo ou igual
jae	Desvia acima ou igual	jnb	Desvia se não for abaixo
jb	Desvia se abaixo	jnae	Desvia se não for acima ou igual
jbe	Desvia se acima que ou igual	jna	Desvia se não for acima

Exemplo anterior com instrução *cmp*:

```
; if initial < 'e'
if01:     cmp initial,'e'
          jae endif01
          inc count
endif01: nop
```


28/44

Operações Lógicas e Caracteres

Operador *not*, negação (!):

Em C:

```
if (!(x ==1))
   y++;
```

Em *assembly*:

```
.if !(x == 1)
inc y
.endif
```

Usando a instrução *cmp*, considere $!(x == 1)$ como $x! = 1$:

```
; if !(x ==1)
if02:     cmp x,1
          je endif02
then02:   inc y
endif02: nop
```


29/44

Operações Lógicas e Caracteres

Operador *or*, ou lógico (||):

Em C:

```
if (x == 1 || y == 2)
   z++;
```

Em *assembly*:

```
.if x == 1 || y == 2
inc z
.endif
```

Usando a instrução *cmp*, se $x == 1$ for verdade, não é preciso checar $y == 2$:

```
; if x == 1 or y == 2
if03:     cmp x,1
          je then03
          cmp y, 2
          jne endif03
then03:   inc z
endif03: nop
```


30/44

Operações Lógicas e Caracteres

Operador *and*, e lógico (&&):

Em C:

```
if (x == 1 && y == 2)
   z++;
```

Em *assembly*:

```
.if x == 1 && y == 2
inc z
.endif
```

Usando a instrução *cmp*, se $x == 1$ for falso, não é preciso checar $y == 2$:

```
; if x == 1 or y == 2
if04:     cmp x,1
          jne endif04
          cmp y, 2
          jne endif04
then04:   inc z
endif04: nop
```


31/44

Operações Lógicas e Caracteres

- O operador *not* (!) tem a maior precedência.
- O operador *and* (&&) tem precedência maior que o *or* (||).
- Parênteses podem ser usados para alterar a precedência.
- No caso de empate, a ordem de avaliação é da esquerda para a direita.

32/44

Operações Lógicas e Caracteres

Em C:

```
if (w == 1 || x == 2 &&  y == 3)
    z++;
```

Usando instruções *cmp*:

```
; if w == 1 || x == 2 && y == 3
if05:    cmp x,2
         jne or05
         cmp y,3
         je then05
or05:    cmp w,1
         jne endif05
then05:  inc z
endif05: nop
```

Em *assembly*:

```
.if w == 1 || x == 2 && y == 3
inc z
.endif
```


33/44

Operações Lógicas e Caracteres

Regras de De Morgan:

not (x and y)	=	not x or not y
not (x or y)	=	not x and not y

```
if (!(x == 1 || y == 1))
   z++;
```

```
.if !(x == 1 || y == 1)
inc z
.endif
```

```
if (!(x==1) && !(y==1))
   z++;
```

```
.if !(x == 1) && !(y == 1)
inc z
.endif
```


34/44

Operações Lógicas e Caracteres

Transformando $!(x == 1)$ em $x! = 1$ fica fácil transformar na versão sem diretivas:

```
if (x != 1 || y != 1)
   z++;
```

```
.if x != 1 && y !=1
inc z
.endif
```

```
if06:    cmp x,1
         je endif06
         cmp y,1
         je endif06
then06:  inc z
endif06: nop
```


35/44

Expressões Aritméticas em Diretivas

- Em C, C++, valores numéricos podem ser usados para representar verdadeiro (não 0) ou falso (0).
- O seguinte código é válido em C, C++:

```
if (x - 1)
    y++;
```

- O código poderia ser reescrito como:

```
if (x != 1)
    y++;
```

- O código a seguir está errado:

```
.if x - 1
inc y
.end
```

- No lugar de usar o valor, é calculado o endereço de memória de x menos 1.

36/44

Expressões Aritméticas em Diretivas

- Nós vimos que a execução de uma instrução *cmp* altera *bits* em *eflags*.
- Instruções aritméticas também alteram *eflags*.
- Na adição, subtração, incremento e decremento as *flags* zero (ZF) e sinal (SF) são alteradas.
- Ao calcular $x - 1$, se x for 1, $x - 1$ é igual a 0 e ZF recebe 1.
- Se o resultado for negativo ou positivo, ZF recebe 0 e SF pode receber 0 ou 1.

37/44

Expressões Aritméticas em Diretivas

- O operador *ZERO?* retorna 1 se a *flag* zero estiver configurada, 0 caso contrário.

```
mov eax, x
dec eax
.if (!ZERO?)
inc y
.endif
```

- Lembrando que o valor de x não deve ser alterado.

38/44

Expressões Aritméticas em Diretivas

Instrução	Descrição	*Flag*	Instrução	Descrição	*Flag*
jc	desvio se *carry*	CF = 1	jnc	desvio se não *carry*	CF = 0
jp	desvio se paridade	PF = 1	jnp	desvio se não paridade	PF = 0
jz	desvio se zero	ZF = 1	jnz	desvio se não zero	ZF = 0
js	desvio se sinal	SF = 1	jns	desvio se não sinal	SF = 0
jo	desvio se *overflow*	OF = 1	jno	desvio se não *overflow*	OF = 0

Podemos agora implementar sem diretivas:

```
if07:    mov eax,x
         dec eax
         jz endif07
then07:  inc y
endif07: nop
```


39/44

Programa Completo

- Um programa que recebe um valor de voltagem e indica se o mesmo é alto, baixo ou aceitável.
- O programa foi feito pensado no sudeste:
  - Voltagem abaixo de 109V é considerada baixa.
  - Entre 110V e 120V é aceitável.
  - Acima de 121V é alta.

40/44

Programa Completo

```c
#include <stdio.h>

int main()
{
    int voltage;
    printf("%s", "Enter an AC Voltage: ");
    scanf("%d", &voltage);
    if (voltage >= 110 && voltage <= 120)
        printf("\n%s\n", "Voltage is Acceptable");
    else
    {
        printf("\n%s\n", "Warning!");
        if (voltage < 110)
            printf("%s\n","Voltage too Low");
        else
            printf("%s\n", "Voltage too High");
    }
    printf("\n");
    return 0;
}
```


41/44

Programa Completo

```
        .686
        .model  flat, c
        .stack  100h
printf  PROTO   arg1:Ptr Byte, printlist:VARARG
scanf   PROTO   arg2:Ptr Byte, inputlist:VARARG
        .data
in1fmt  byte    "%d",0
msg1fmt byte    "%s",0
msg2fmt byte    0Ah,"%s",0Ah,0
msg4fmt byte    "%s",0Ah,0
msg6fmt byte    0Ah,0
msg1    byte    "Enter an AC voltage: ",0
msg2    byte    "Voltage is Acceptable",0
msg3    byte    "Warning!",0
msg4    byte    "Voltage too Low",0
msg5    byte    "Voltage too High",0
voltage sdword  ?
```


42/44

Programa Completo

```
        .code
main    proc
        INVOKE printf, ADDR msg1fmt, ADDR msg1
        INVOKE scanf, ADDR in1fmt, ADDR voltage
        .if voltage >= 110 && voltage <= 120
        INVOKE printf, ADDR msg2fmt, ADDR msg2
        .else
        INVOKE printf, ADDR msg2fmt, ADDR msg3
        .if voltage < 110
        INVOKE printf, ADDR msg4fmt, ADDR msg4
        .else
        INVOKE printf, ADDR msg4fmt, ADDR msg5
        .endif
        .endif
        INVOKE printf, ADDR msg6fmt
        ret
main    endp
        end
```


43/44

Resumo

- Se possível, evite instruções com *nots* (por exemplo, *jg* é melhor que *jnle*).
- Quando estiver implementando sentenças *ifs* sem diretivas, o desvio condicional geralmente precisa ser invertido.
- Estruturas *if-then-else-if* aninhadas são mais legíveis que estruturas *if-then-if*.
- MASM não tem *switch*.
- A diretiva *.elseif* pode dificultar a alteração do código.
- Bons nomes para rótulos ajudam na legibilidade.
- Ao comparar caracteres, use desvios sem sinal.
- Lembre da precedência dos operadores lógicos.
- Evite usar expressões aritméticas com diretivas.

44/44