instruções aritméticas

Instruções de comparação
instruções de controle de fluxo

1

instruções aritméticas

Instruções básicas

ADICIONE *destino, fonte;* (destino=destino+origem)

destino e *origem* têm as mesmas regras de mov

destino INC ; (destino=destino+1)

INC é mais rápido que ADD

SUB *destino, origem* (destino=destino-origem)

destino e *origem* têm as mesmas regras de mov

destino DEC ; (destino=destino-1)

DEC é mais rápido que SUB

2

Instrução de comparação

CMP *destino, origem* (temporário=destino-origem)

destino e *fonte* têm as mesmas regras que mov

destino é inalterado

FLAGS são afetados de acordo com a operação de subtração

Nota: *O registrador Flag é um registrador de finalidade especial. Dependendo do valor do resultado após qualquer operação aritmética e lógica os bits de flag torne-se definido (1) ou reinicializado (0).*

3

Instruções de controle de fluxo
salto incondicional

etiqueta JMP

Executa um salto incondicional para o rótulo o rótulo pode ser colocado antes ou depois da linha jmp o rótulo deve ser único no código do arquivo

Exemplo

```
próximo: ADICIONE EAX,2
         JMP próximo
```

Este código executa um loop infinito

4

Saltos condicionais de instruções de controle de fluxo

Os saltos condicionais controlam o programa de acordo com as condições

O salto é feito apenas se a condição for verificada

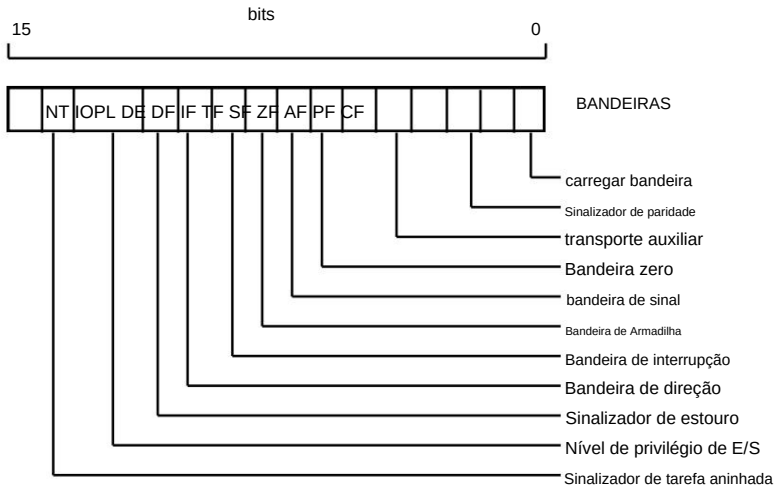
Se o salto não for feito, a próxima instrução de linha será executada

A condição é verificada pelos contendores de FLAGS

Os saltos condicionais são afetados por qualquer instrução que altere os FLAGS

Existem condições diferentes para números assinados ou não assinados

5



ET TS EMMP PE PALAVRA DE ESTADO DA MÁQUINA

6

Proteção Habilitada

Monitorar Coprocessador

Emular Coprocessador
Tarefa trocada

Extensão do processador

Ponteiro de Instrução IP

Instruções de controle de fluxo

saltos condicionais (mais usados no vermelho)

Descrição do código de operação		Sinalizadores de CPU
JA	Acima	CF = 0 e ZF = 0
JAE	Acima ou igual	FC = 0
JB	Abaixo	CF
JBE	Abaixo ou igual	CF ou ZF
JC	Carregar	CF
JE	Igualdade	ZF
JG	Maior(es)	ZF = 0 e SF = DE
JGE	Maior de igual(es) SF = OF	
JL	Menos(ns)	SF ÿ OF
JLE	Menos igual(es)	ZF ou SF ÿ OF
JNA	Não acima	CF ou ZF
JNAE	Nem acima nem igual	CF
JNB	Não abaixo	FC = 0
JNBE	Nem abaixo nem	CF = 0 e ZF igual = 0

Descrição do código de operação		Sinalizadores de CPU
JNC	não carrega	FC = 0
JNE	Não igual	ZF = 0
JNG	Não maior	ZF ou SF ÿ OF
JNGE	Nem maior nem igual	SF ÿ OF
JNL	Não menos	SF = DE
JNLE	Nem menos nem igual	ZF = 0 e SF = DE
JNO	Não estouro	DE = 0
JNP	Sem paridade	FP = 0
JNS	Não negativo	SF = 0
JNZ	diferente de zero	ZF = 0
JO	Estouro(s)	DE
JP	Paridade	PF
JPE	Paridade	PF
JPO	Sem paridade	FP = 0
js	Negativo(s)	SF
JZ	Nulo	ZF

7

Instrução	Descrição	Sinalizadores testados
JE/JZ	Salto Igual ou Salto Zero	ZF
JNE/JNZ	Salto não igual ou salto diferente de zero	ZF
JG/JNLE	Salte Maior ou Salte Não Menos/Igual	OF, SF, ZF
JGE/JNL	Salte Maior ou Não Salte Menos	OF, SF
JL/JNGE	Saltar menos ou não maior/igual	OF, SF
JLE/JNG	Salto menor/igual ou salto não maior	OF, SF, ZF

8

Instrução	Descrição	Sinalizadores testados
JE/JZ	Salto Igual ou Salto Zero	ZF
JNE/JNZ	Salto não igual ou salto diferente de zero	ZF
JA/JNBE	Salte acima ou não abaixo/igual	CF, ZF
JAE/JNB	Pular acima/igual ou não pular abaixo	CF
JB/JNAE	Salte Abaixo ou Não Salte Acima/Igual	CF
JBE/JNA	Pular abaixo/igual ou não pular acima	AF, CF

9

Instrução	Descrição	Sinalizadores testados
JXCZ	Salte se CX for zero	nenhum
JC	Pular Se Carregar	CF
JNC	Pular se não carregar	CF
JO	Saltar se estourar	DE
JNO	Saltar se não houver estouro	DE
JP/JPE	Paridade de salto ou Paridade de salto par	PF
JNP/JPO	Salto Sem Paridade ou Salto Paridade Ímpar	PF
jS	Sinal de Salto (valor negativo)	SF
JNS	Salto sem sinal (valor positivo)	SF

10

Implementação de loop de instruções de controle de fluxo

```
        mov ecx,0                    ;inicia o contador de loop
repita: …
        …
        incl ecx
        cmp ecx,100 jne
        repetição
        … (a próxima linha é executada quando ecx atinge 100)
```

Código de linhas de loop executado 100 vezes

11

Implementação de loop de instruções de controle de fluxo

```
        mov ecx,100; iniciar o contador no valor máximo
repita: …
        …
        dezembro ecx
        jnz repita
        não é necessária … (a próxima linha é executada quando ecx atinge 0)
```

Código de linhas de loop executado 100 vezes

a instrução dec altera os sinalizadores a instrução cmp

12

Instruções de controle de fluxo se implementação

código C

```
if(x>0){
        x--;
}
y=x;
```

código de montagem

```
        mover eax, x
        cmp eax,0
        jle end_if ;salto menor ou igual (condição para não se)
        dec eax ;if código
end_if: mov y,eax ;coloque o valor de x (eax) em y
```

13

Instruções de controle de fluxo if then else implementação

código C

```
if(x>0){
    x--;

} outro{
    x++;
}
y=x;
```

código de montagem

```
        mover eax, x
        cmp eax,0
        jle mais        ;salto menor ou igual (condição para outro)
        dezembro exax   ;se código
        jmp end_if      ;este salto impede a execução do código
                        else quando o código if é executado
senão: inc eax          ;outro código
end_if: mov y,eax       ;coloque o valor de x (eax) em y
```

Códigos if e else são mutuamente exclusivos

14

Instrução MUL/IMUL

Existem duas instruções para multiplicar dados binários.

1. A instrução MUL (Multiply) lida com dados não assinados 2. A instrução IMUL (Multiplicação Integer) lida com dados assinados.

Ambas as instruções afetam o flag Carry e Overflow.

15

instruções aritméticas

fonte MUL

MUL multiplica inteiros **sem sinal**

Ficava apenas no parâmetro *(source)* que pode ser memória ou registrador

Os outros parâmetros são fixos

- 2 ÿ Multiplicando • × 8

ÿ Multiplicador

• 16 ÿ Produto

16

Quando dois bytes são multiplicados

• O multiplicando está no registrador AL • O

multiplicador é um byte na memória ou em outro registrador. • O produto

está em AX. • Os 8 bits

de ordem superior do produto são armazenados em AH e os 8 bits de ordem inferior são armazenados em AL.

17

Quando dois valores de uma palavra são multiplicados

O multiplicando deve estar no registrador AX.
O multiplicador é uma palavra na memória ou outro registrador.

MUL DX, você deve armazenar o multiplicador em DX e o multiplicando em AX.

O produto resultante é uma palavra dupla, que precisará de dois registradores.
A parte de ordem superior (mais à esquerda) é armazenada em DX e a parte de ordem inferior (mais à direita) é armazenada em AX.

18

Quando dois valores de doubleword são multiplicados

- O multiplicando deve estar em EAX • O multiplicador é um valor doubleword armazenado na memória ou em outro registrador.
- O produto gerado é armazenado nos registradores EDX:EAX • Os 32 bits de ordem superior são armazenados no EDX • Os 32 bits de ordem inferior são armazenados no registrador EAX.

19

instruções aritméticas

Exemplos

```
MOV AL, 10
MOV BL,5
MUL BL; AX=AL*BL=10*5=50

MOV AX,100
MOV BX,50
MUL BX; DX:AX=100*50=5000

MOV EAX,1000
MOV EBX,100
MUL EBX; EDX:EAX=1000*100=100000
```

20

instruções aritméticas

fonte IMUL

IMUL multiplica números inteiros **com sinal**

Ficava apenas no parâmetro *(source)* que pode ser memória ou registrador

Os outros parâmetros são fixos

21

instruções aritméticas

Exemplos

```
MOV AL, 10
MOV BL,5
IMUL BL; AX=AL*BL=10*5=50

MOV AX,100
MOV BX,-50
IMUL BX; DX:AX=100*(-50)=-5000

MOV EAX,-1000
MOV EBX,100
IMUL EBX; EDX:EAX=(-1000)*100=-100000
```

22

instruções aritméticas

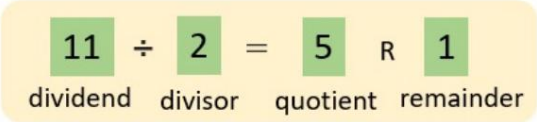
fonte DIV

DIV divide inteiros **sem sinal**

Ficava apenas no parâmetro *(source)* que pode ser memória ou registrador

Os outros parâmetros são fixos

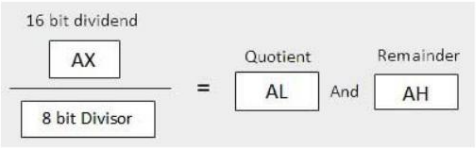
Para cada situação se o resultado não couber em AL, AX ou EAX, uma exceção é gerada e o programa para.



23

Quando o divisor é 1 byte

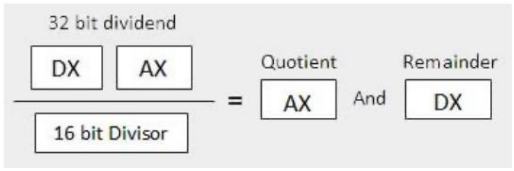
- Presume-se que o dividendo esteja no registrador AX (16 bits).
- Após a divisão, o quociente vai para o registrador AL e o resto vai para o registrador AH.



24

Quando o divisor é 1 palavra

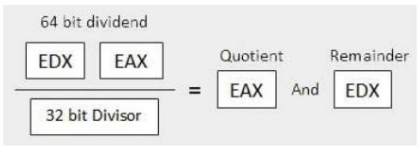
- Presume-se que o dividendo tenha 32 bits de comprimento e esteja nos registradores DX:AX.
- Os 16 bits de ordem superior estão em DX
- Os 16 bits de ordem inferior estão em AX.
- Após a divisão, o quociente de 16 bits vai para o registrador AX e o resto de 16 bits vai para o registrador DX.



25

Quando o divisor é doubleword

- Presume-se que o dividendo tenha 64 bits de comprimento e esteja nos registradores EDX:EAX.
- Os 32 bits de ordem superior estão em EDX e os 32 bits de ordem inferior estão em EAX.
- Após a divisão, o quociente de 32 bits vai para o registrador EAX e o resto de 32 bits vai para o registrador EDX.



26

instruções aritméticas

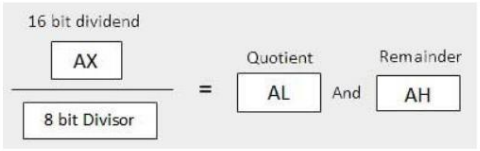
Exemplos

MOV AX, 102

MOV BL,5

DIV BL AL=AX/BL=102/5=20

AH=AX%BL=102%5=2



27

instruções aritméticas

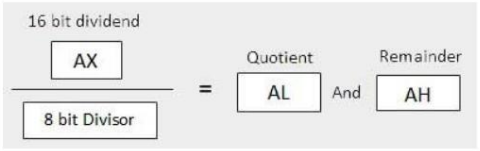
Exemplos

MOV AX,1001

MOV BL,2

DIVBL;

Esta instrução produz um erro de tempo de execução porque o resultado é maior que 255



28

instruções aritméticas

Solução para dividir 1001 por 2

```
MOV AX,1001
MOVDX,0
MOV BX,2
DIV BX
```

Neste caso a fonte (BX) é de 16 bits, e o resultado é

AX=DX:AX/BX=1001/2=500

DX=DX:AX%BX=1001%2=1

29

instruções aritméticas

fonte IDIV

IDIV divide inteiros **com sinal**

Ficava apenas no parâmetro *(source)* que pode ser memória ou registrador

Os outros parâmetros são fixos

Para cada situação se o resultado não couber em AL, AX ou EAX, é gerada uma exceção e o programa para.

30

instruções aritméticas

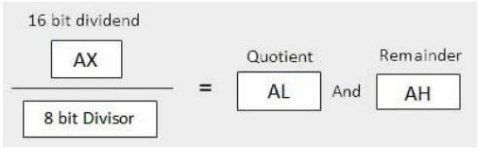
Exemplos

MOV AX,401

MOV BL,2

IDIV BL;

Esta instrução produz um erro de tempo de execução porque o resultado é maior que 127



31