Add:

* addb -> adiciona bytes (8bits)
* addw -> adiciona words (16bits)
* addl -> adiciona double words (long) (32 bits)
* addq -> adiciona quad words (64bits)

A conversão de tipo tem prioridade mais alta que as operações aritméticas

A passagem de um tipo de dados mais pequeno para outro maior irá produzir:

extensão com zeros, para números sem sinal

extensão de sinal, para números com sinal

As instruções que movam ou produzam valores para registos de 32 bits, vão tmb mudar os 32 bits mais significativos para 0

**add origem, destino**

permite adicionar dois inteiros

destino = destino+origem

origem pode ser endereço de memoria, valor constante ou registo

destino pode ser um endereço de memória ou um registo

endereços de memória não podem ser usados para origem e destino numa única instrução

addb $10, %al -> adiciona 10 ao registo al (8bits) : al = al + 10

addw %bx, %cx : cx = cx + bx

addl var1%(rip), %eax : eax = eax + var1

addl %eax, %eax : eax = eax + eax

addq %rcx, %rax : rax = rax + rcx

ISTO É TUDO COM REGISTOS

**sub origem, destino**

Vai fazer destino – origem

Mesma lógica do add

**inc destination**

**dec destination**

incrementa e decrementa numa unidade: é o típico ++ ou –

descb %cl -> cl = cl -1

mesma lógica

**adc origem, destino**

pode ser usada para adicionar dois nrs inteiros com o valor contido na carry flag, que resultou de uma adição anterior

1. <--------------- este é o carry flag

1 7

+ 2 8

-------------------------------

4 5

Ex: se for por exemplo 6+2=8<10, o carry será 0

Se o carry flag estiver !=0 a conta está errada, pq não houve espaço para fazer a conta

No adc estamos a fazer a conta da zona da coluna do carry

Só depois de fazer um add é que se faz um adc

O adc vai buscar a última operação

Realiza a operação destino = destino + origem + CF(carry flag)

Só se usa em operações sem sinal

Tmb tem as 4 variantes do b , w, l e q

movb $0xFF, %al

movb $0x1, %ah

movb $0x1, %bl

movb $0x0, %bh

#somar 2 nrs de 16 bits usando instruções de 8 bits

addb %al, %bl #bl= bl + al (somar os 8 bits menos significativos)

adcb %ah, %bh #bh = bh + ah + CF (somar os 8 bits mais significativos com o carry)

usando esta técnica conseguimos somar nrs em 128 bits se usarmos o addq e adcq

**sbb origem, destino**

3 2

- (1) 7 <--------- (1) é o borrow

-------------------------------

4 5

é a mesma lógica do adc

destino = destino – (origem + CF)

NA MULTIPLICAÇÃO UM DOS ELEMENTOS TEM DE ESTAR OBRIGATORIAMENTE NO REGISTO rax

O RESULTADO É ARMAZENADO NOS REGISTOS rdx (64 bits mais significativos) E NO REGISTO rax (64 bits menos significativos)

MULTIPLICAÇÃO SEM SINAL: **mul origem**

Multiplica dois inteiros sem sinal

Destino = origem \* operando2

O operando2 terá de ser registo al, ax, eax ou rax

O destino terá de ser registo ax, dx:ax, edx:eax ou rdx:rax

Tamanho de origem Operando2 Destino

8 bits al ax

16 bits ax dx:ax

32 bits eax edx:eax

64 bits rax rdx:rax

dx:ax = cx \* ax -> 32 bits pq tem 2 registos de 16

COM SINAL: **imul origem**

Multiplica dois inteiros com sinal

Destino = destino \* origem

(mesma tabela de cima)

DIVISAO COM E SEM SINAL EXISTE APENAS COM INSTRUÇÕES DE 1 ÚNICO OPERANDO

dividendo | divisor

|---------------

resto | quociente

produz quociente e resto, nunca produz um float

QUANDO O DIVIDENDO FOR UM VALOR REPRESENTADO EM 64 BITS

O VALOR DEVE SER ARMAZENADO EM rax

OS BITS DE rdx DEVEM SER COLOCADOS DE UMA DAS DUAS FORMAS:

- TODOS A ZERO, PARA ARITMÉTICA SEM SINAL

- COM A EXPANSÃO DO BIT DE SINAL DE rax, PARA ARITMÉTICA COM SINAL

Dividendo -> rdx:rax -> a parte do rdx tem de ser tratada !!!!

**div divisor**

**idiv divisor**

div é para sem sinal

idiv é para com sinal

Tamanho de divisor Dividendo quociente resto

8 bits ax al ah

16 bits dx:ax ax dx

32 bits edx:eax eax edx

64 bits rdx:rax rax rdx

INICIALIZAR TODOS OS BITS DO DIVIDENDO

Extensão de sinal com registos específicos:

cbtw (preferível, pq o outro tem mais variações)) ou cbw -> estensao de sinal do byte no registo al para word, colocando no registo ax

usamos para não cometer o tal erro e fazer a extensão do sinal de eax para edx:eax

se o dividendo for o resultado de uma multiplicação já com 128 bits, ao fazer a divisão ele já fica bem e estaremos a estragar o resultado ao fazer a extensão de sinal