OPERAÇÕES AO NIVEL DO BIT

- são usadas para manipular números binários

- cada bit é comparada individualmente utilizando-se a função logica adequada

Funções logicas:

- AND

- OR

- XOR

- NOT

AND

Operador & do C

X Y AND

0 0 0

0 1 0

1 0 0

1 1 1

amd origem, destino

destino = destino AND origem (resultado em destino)

origem – endereço de memoria, valor constante ou registo

destino – endereço de memoria ou registo

OR

Operador | do C

X Y OR

0 0 0

0 1 1

1 0 1

1 1 1

Or origem, destino

Destino = destino OR origem

XOR

Operador ^ do C

X Y XOR

0 0 0

0 1 1

1 0 1

1 1 0

Xor origem, destino

Destino = destino XOR origem

NOT

Operador ~ do C

X NOT

0 1

1 0

Not destino

Destino = NOT destino

A INSTRUÇO NEG MODIFICA O SINAL DE UM INTEIRO

O OPERADOR ! NÃO É UM OPERADOR AO NIVEL DO BIT, PQ CONSIDERA O VALOR INTEIRO COMO UM VALOR LOGICO

X NOT X NEG X !X

0 -1 0 1

-1 0 1 0

1 -2 -1 0

0000(2) = 0(10)

1111(2) = -1(10)

0001(2) = 1(10)

1110(2) = -2(10) -> 1110(2) -> inv -> 0001(2) -> +1 -> 0010(2) = 2, e como 1110 começa por 1 então é -2

DESLOCAMENTO DE BIT PARA A ESQUERDA

00000011 passa a = 3(10)

0(carry) 00000110 = 6(10)

DESLOCAMENTO DE BIT PARA A DIREITA

00000110 passa a = 6(10)

00000011 0(carry) = 3(10)

Shl / sal

Operador << em C

Deslocamento de bits para a esquerda

Shl e sal são operações equivalentes

As duas existem apenas por motivo de consistência

Zeros entram do lado direito e o ultimo bit a sair (do lado esquerdo) fica na carry flag CF

Ex: fazer 2 vezes o sal -> o nr fica 4x superior

Shl destino (ou sal destino)

Destino = destino \* 2

Desloca 1 vez

Shl %cl, destino (ou sak %cl, destino)

Destino = destino \* 2^cl

Desloca cl vezes

Shl $n, destino (ou sal $n, destino)

Destino = destino \* 2^n

Como o de cima, mas é n vezes

REALIZAR UM DESLOCAMENTO NUM NUM INTEIRO COM SINAL PODE ALTERAR O SINAL DO NUMERO

DESTINÇAO ENTRE AS DA DIREITA:

Shr – deslocamento de bits logico, não preserva o sinal

Sar – deslocamento de bits aritmético, preserva o sinal

Shr

Operador >> em C, quando aplicado a números sem sinal

Zeros entram do lado esquerdo e o ultimo bit a sair (do lado direito) fica na carry flag CF

Não preserva o sinal do numero

Shr destino

Destino = destino / 2

Shr %cl, destino

Destino = destino / 2^cl

Cl vezes

Shr $n, destino

N vezes

Sar

Operador em C, quando aplicado a números com sinal

Bits de valor igual ao bit de sinal (bit + significativo) entram do lado esquerdo e o ultimo bit a sair ( do lado direito) fica na carry flag CF

Preserva o sinal do numero

Sar destino

Destino = destino / 2

Sar %cl, destino

Cl vezes

Sar $n, destino

N vezes

Rol / ror – ROTAÇAO DE BITS

NÃO EXISTE OPERADOR EQUIVALENTE EM C, tem de ser feito criando uma função para tal

Rol – rotação de bits para a esquerda

O bit mais à esquerda passa para o lugar mais à direita

Rol destino

1 rotaçao para a esquerda

Rol %cl, destino

Cl rotações

Rol $n, destino

N rotaçoes

Ror – rotação de bits para a direita

O bit mais à direita passa para o lugar mais à esquerda

Ror destino

1 rotaçao para a direita

Ror %cl, destino

Cl rotações para a direita

Ror $n, destino

N rotações para a direita

Os bits que saem numa extremidade, entram na outra

Rcl / rcr

Não existe operador equivalente em c

Rotação de bits para a esquerda com carry

É o mesmo que o rol e ror, mas considera também o carry como se fosse um dos bits do byte (no sentido em que também empurra para ele e também é empurrado dele)

MASCARAS

Padrao de bits que seleciona um conjunto de bits num numero

Ex: al tem valor 00101100

Queremos os 4 bits menos significativos

Fazemos:

movb $0b00101100, %al

movb $0b00001111, %ah

andb %ah, %al #al = 0b0001100

o que queremos deixar passar colocamos 1

o que queremos reter colocamos 0

ex: queremos substituir os 4 menos significativos de al pelos 4 menos significativos de cl

movb $0b00101100, %al

movb $0b01000011, %cl

movb $0b11110000, %ah

andb %ah, %al #al = 0b00100000

notb %ah #ah = 0b00001111 (inverter a mascara)

andb %ah, %cl #cl = 0b00000011

orb %cl, %al #al = 0b00100011

xor pode ser usada para definir um numero como sendo zero ou para comparar dois números

o resultadp de uma operação xor entre dois números iguais é zero

xor %ebx, %ecx #se ebx == ecx a ZF está ativada

jz is\_equal

xor tmb serve para inverter os bits de um registo

0 onde se pretende manter o bit original

1 onde se pretende inverter o bit original

Movb $0b00101100, %al

Movb $0b00001111, %ah

Xorb %ah, %al (? Não consegui ver)

AND

Colocar bits a 0

Obter o valor de alguns bits

Arredondar a potencia de base 2

Obter resto de divisão por potencia de base 2

OR

Colocar bits a 1

Juntar bits de 2 num

XOR

Verificar se 2 num são iguais (? Nums ou bits não vi bem)

#define mask(n) ((1<<(n))-1)

#define mask2(n1, n2) (mask(n2-n1+1)<<(n1))

#include <stdio.h>

Int main(void){

Printf(“%d\n”, mask(2));

Printf(“%d\n”, mask(3));

Printf(“%d\n”, mask2(1,2));

Printf(“%d\n”, mask2(1,3));

Return 0;

}

Imprime 3,7,6,14