**rflags – registo que representa valores booleanos**

0 – inativo/falso

1 – ativo/verdadeiro

flags mais importantes:

CF – carry flag (bit 0)

Ativada quando existe transporte (Carry) ou empréstimo (borrow) do bit mais significativo

Inativa caso contrário

((aritmética sem sinal))

ZF – zero flag (bit 6)

Ativada quando o resultado da operação é zero

Inativa caso contrário

SF – sign flag (bit 7)

Toma o valor do bit mais significativo (bit de sinal) do resultado:

0, se positivo

1, se negativo

OF – overflow flag (bit 11)

Ativada quando o resultado da operação é demasiado grande (nr positivo) ou demasiado pequeno (nr negativo) para ser armazenado no operando de destino

Inativa caso contrario

((aritmética com sinal))

**Rip – registo que indica o endereço de memoria da próxima instrução a ser executada**

Após a execução da instrução, o rip fica automaticamente com o endereço da próximo instrução

Uma instrução de salto pode continuar a execução do programa numa posição do programa completamente diferente:

Salto incondicional

Salto condicional

Salto incondicional

jmp endereço – modifica o valor do registo rip para endereço, numa zona no programa para onde deve continuar a execução do programa, usualmente indicada por uma etiqueta

*jmp end*

*….*

*end*

não faz o ….

Salto condicional

Os saltos condicionais são ou não efetuados, dependendo do estado do registo rflags aquando da execução da instrução de salto

je – jump if equal (2Z=1)

jl – jump if less (SF<>OF)

jg – jump if greater (2F=0 e SF=OF)

IMPORTANTE: ANTES DE UM SALTO CONDICIONAL, OS CODIGOS DE CONDIÇAO DO REGISTO RFLAGS TÊM DE SER DEFINIDOS DE UMA FORMA APROPRIADA POR UMA OPERAÇÃO

**INSTRUÇAO DE COMPARAÇAO**

cmp operando1, operando2

o operando2 é que é comparado em relação ao operando1 !!!

é a forma mais habitual de avaliar dois valores para um salto condicional

compara o segundo com o primeiro, através de uma subtração: operando2 – operando1

não modifica os operandos, mas os códigos de condição do rflag

se:

operando2==operando1 -> 2Z=1

operando2>operando1 -> SF=0

operando2<operando1 -> SF=1

pode ser aplicada a operandos de 8(b), 16(w), 32(l) ou 64(q) bits

Operador Descriçao

~ inverte cada um dos bits

^ XOR

& AND

| OR

PARA VER SE UMA COISA É MAIOR QUE OUTRA E FAZER O SALTO:

jg se tem sinal

ja se não tem sinal

+ jae – above or equal

+ jbe – below or equal

(estes dois nao estao no moodle)

*#...*

*cmpq %rcx, %rsi #comprara rsi contra rcx (através de: rsi – rcx)*

*jg jmp\_rsi\_is\_greater*

*je jmp\_rsi\_is\_equal*

*jl jmp\_rsi\_is\_less*

*jmp\_rsi\_is\_greater:*

*movq $1, %rax*

*jmp end*

*jmp\_rsi\_is\_equal:*

*movq $0, %rax*

*jmp end*

*jmp\_rsi\_is\_less:*

*movq $-1, %rax*

*end:*

*ret*

para diminuir o nr de instruções, podemos retirar o greater, pq se não for greater ele vai saltá-lo. Se não for, ele faz

**EM C:**

*long x;*

*long y;*

*long test\_xy(){*

*if(x>y){*

*return 1;*

*} else {*

*Return 0;*

*}*

*}*

**EM ASSEMBLY:**

*test\_xy:*

*movq x(%rip), %rdi #rdi=x*

*cmpq y(%rip), %rdi #x?y*

*jle false #se x<=y salta*

*movq $1, %rax #rax=1*

*jmp end*

*false:*

*movq $0, %rax #rax=0*

*end:*

*ret*

OPERAÇÕES ARITMÉTICAS: DETETAR CARRY

jc endereço – jump if carry – efetua o salto para o endereço se a flag do carry estiver ativa (CF=1)

SEM SINAL

A flag de carry é ativada quando o resultado prigina transporte (carry) ou empréstimo (borrow) no bit mais significativo

OPERAÇÕES ARITMÉTICAS: DETETAR OVERFLOW

COM SINAL

jo – jump is overflow – salta se a flag estiver ativa (OF=1)

é ativada quando o resultado é um nr positivo demasiado grande, ou negativo demasiado pequeno, para ser armazenado no registo

OPERAÇÕES ARITMÉTICAS: DETETAR CARRY E OVERFLOW

Em c não temos acesso a estas flags, os overflows não são sinalizados como erros

Podemos verificar se ocorreu carry em x+y se e so se sum<x (ou sum<y) para uma adição sem sinal

Para adições com sinal, o calsulo de soma terá:

Underflow negativo, se e só se: x<0 e y<0 mas sum>=0

Overflow positivo, se e só se: x>0 e y>0 mas sum<=0

((no moodle, o exemplo não está de acordo com o que aqui diz))

Em código maquina não existem instruções como o do-while, while e for

Para tal, são usadas combinações de testes condicionais e ciclos

((A FORMA DE FAZER OS “CICLOS” EM ASSEMBLY ESTÁ NO MOODLE MAIS EXPLICADA))

DO WHILE:

my\_loop:

…

jg my\_loop

WHILE:

Fact\_while:

…

Jle end\_my\_loop

…

Jmp my loop

End\_my\_loop:

ret

FOR: (é convertido a while e feito da forma do while)

INSTRUÇOES DE LOOP:

Loop, loope, loop, loopne, loopnz

Loop etiqueta:

decq %rcx

jnz etiqueta

LOOPS TENTAM FLAGS MAS NÃO MODIFICAM