“Run to the Hills.” (Maiden, I.)

Multiplicação e Divisão de Inteiros no x86-64

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Multiplicação sem sinal

- Utilize a instrução ***mul[bwlq]*** para efetuar uma multiplicação de **valores sem sinal**

  mul[bwlq] FONTE

  - Onde FONTE é um registrador ou uma posição da memória, e representa o multiplicador
  - O valor a ser multiplicado **deve estar em rax** (ou eax, ou ax ou al)
  - O resultado é armazenado em rax, ou em rax e rdx, dependendo do tamanho dos operandos
  - Ambos operandos **devem ter o mesmo tamanho**

Multiplicação sem sinal

Instrução	Tam. Operandos	Reg. A	Parte alta do resultado	Parte baixa do resultado
mulb	8 bits	al	ah	al
mulw	16 bits	ax	dx	ax
mull	32 bits	eax	edx	eax
mulq	64 bits	rax	rdx	rax

Multiplicação com sinal

- Para valores com sinal (complemento de dois) podemos utilizar a instrução imul[bwlq]
  - Três variantes

    imul FONTE
    - **Mesma lógica do mul**

    imul FONTE, DESTINO
    - Fonte pode ser um reg., posição na memória ou imediato. Destino deve ser um registrador
    - DESTINO = DESTINO * FONTE
    - Se o resultado é muito grande, **o valor é truncado**

    imul IMEDIATO, FONTE, DESTINO
    - Fonte pode ser um reg. ou posição na memória. Destino deve ser um registrador
    - DESTINO = FONTE*IMEDIATO
    - Se o resultado é muito grande, **o valor é truncado**

Exercício

1. Crie uma função em *assembly* do x86-64 que recebe um inteiro sem sinal N, e retorna o resultado de N!. Crie um *main* em *C* onde um valor é solicitado ao usuário, e o seu fatorial é impresso na tela. Utilize *extern* em C para importar sua função montada em assembly.
    - Considere que
        - N é um inteiro sem sinal de 4 bytes
        - A resposta cabe em 4 bytes

Divisão sem Sinal

- A divisão sem sinal por ser feita através da instrução ***div[bwlq]***
  div[bwlq] FONTE
  - Onde FONTE é um registrador ou uma posição da memória, e representa o **divisor**
  - O valor a ser dividido **(dividendo)** deve estar em rdx:rax
    - rax com a parte baixa
    - rdx com a parte alta
  - O resultado é armazenado em rax, ou em rax e rdx, dependendo do tamanho dos operandos

Divisão sem Sinal

Instrução	Tam. Divisor	Parte alta Dividendo	Parte baixa Dividendo	Quociente	Resto
divb	8 bits	ah	al	al	ah
divw	16 bits	dx	ax	ax	dx
divl	32 bits	edx	eax	eax	edx
divq	64 bits	rdx	rax	rax	rdx

Podemos dividir um número de até 128 bits!

UDESC

Divisão com Sinal

- Para divisão com sinal, utilize
  idiv[bwlq] FONTE
  - Mesmo conceito da divisão sem sinal
  - Diferente da multiplicação com sinal, não temos outros formatos

Exercício

2. Crie uma função que divide um valor x por um y
   - Pelo bem de nossa sanidade
     - Considere que o dividendo e o divisor ocupam 4 bytes
       - **Como uma primeira solução, carregue o dividendo para eax, e 0 para edx**
     - A função retorna o resultado da divisão, e armazena o resto da divisão em um endereço de memória
     - Equivalente em C:
       int dividir(int dividendo, int divisor, int resto);*
   - Depois, criar um **main em C** que usa essa função

Exercício

Lembre-se que os parâmetros são passados em rdi, rsi, rdx …, mas vamos precisar do rdx no idiv

Copia o dividendo para eax. O dividendo tem apenas 4 bytes, então os 4 bytes mais altos (idivl considera que o dividendo tem 8 bytes) são zerados

```
        .text
        .globl dividir
        .type dividir, @function
dividir:
        pushq %rbp
        movq %rsp, %rbp       #fim do prólogo

        movq %rdx,%rcx #copiando rdx para auxiliar
        movl %edi,%eax #os 4 bytes para eax
        movl $0, %edx     #os 4 bytes mais altos zerados
        idivl %esi            #divisor em %esi
        #o resultado já foi armazenado em eax pelo idivl
        movl %edx, (%rcx) #copia o resto para o endereço em rcx

        movq %rbp, %rsp
        popq %rbp    #fim do epílogo
        ret
```

Exercício

```
#include<stdio.h>

extern int dividir(int dividendo, int divisor, int* resto);

int main(){
	int dividendo, divisor, resultado, resto;
	scanf("%i %i", &dividendo, &divisor);
	resultado = dividir(dividendo, divisor, &resto);
	printf("%d/%d=%d resto %d\n", dividendo, divisor, resultado, resto);

	return 0;
}
```

Montar e fazer a linkedição

as dividir.s -o dividir.o

gcc mainDividir.c dividir.o -o mainDividir

Exercício

- Teste a solução
  - Funciona? Problemas?

Exercício

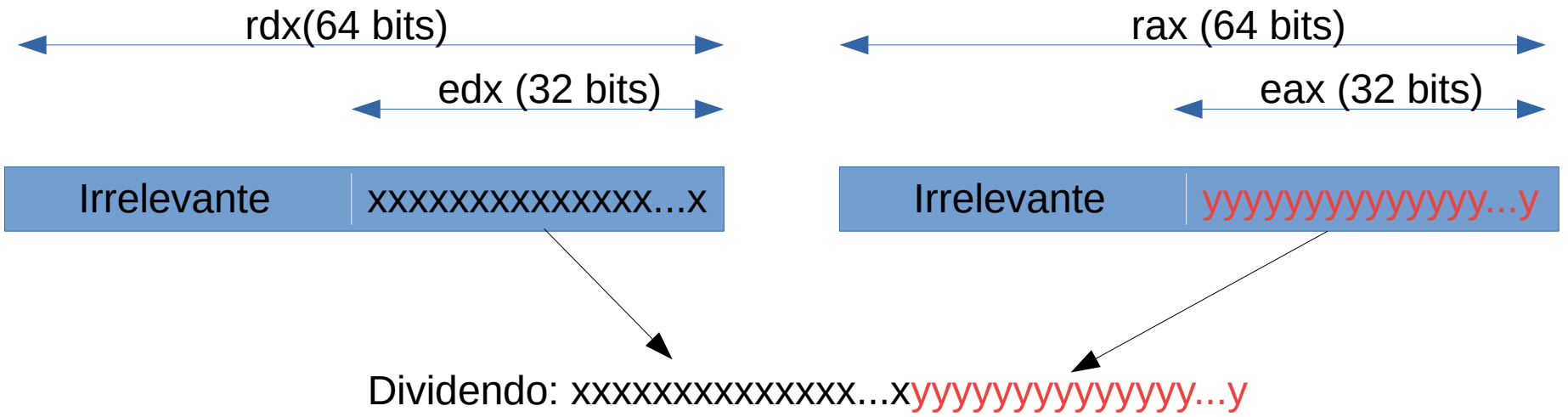
- Teste a solução
  - Funciona? Problemas?
    - **Teste -5/2 e veja que temos um resultado absurdo!**
    - Mas fizemos tudo certo!
      - Carregamos eax e edx
      - Utilizamos divisão com sinal
      - Seguimos a convenção para passagem de parâmetros
      - Deve ser algum problema do C, defeito no hardware, algum espírito maligno, ...
        - Temos que colocar a culpa em alguém

Exercício

- Teste a solução
  - Funciona? Problemas?
    - Teste -5/2 e veja que temos um resultado absurdo!
    - **Discuta com seus colegas e tente encontrar o erro**

Escovando os bits

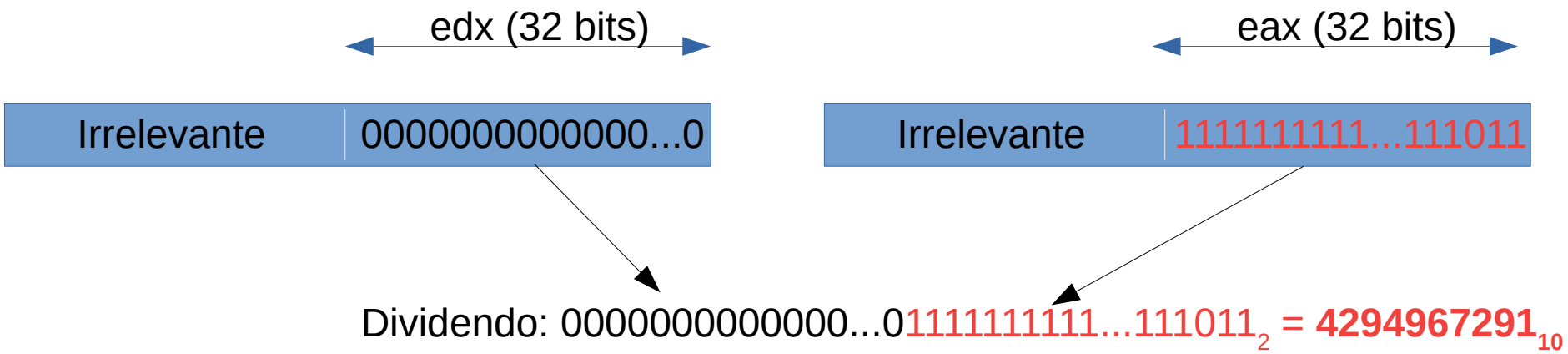
- O div**l** assume que o dividendo pode ter 8 bytes (64 bits)
  - Os 4 bytes mais baixos em **eax**, e os 4 mais altos e **edx**



Escovando os bits

```
movl %edi,%eax #os 4 bytes para eax
movl $0, %edx     #os 4 bytes mais altos zerados
```

- Caso que **não** funciona: -5/2
  - -5_{10} em binário é $11111111111111111111111111111011_2$
    - Considerando-se 32 bits
    - **Complemento de 2**



Complemento de 2

- Em complemento de 2, os bits mais altos são 0 para valores positivos, e 1 para valores negativos
  - “Completamos” com
    - 0’s a esquerda para valores positivos
    - 1’s a esquerda para valores negativos
- Podemos utilizar jumps e operadores binários para definir como completar edx
  - No entanto, dentre as milhares de instruções disponíveis no x86-64, temos instruções para extensão de sinal
  - Procure por cwd no **manual da Intel**

Exercícios

3. Corrija o problema na função de divisão para que valores negativos sejam divididos corretamente

4. Crie uma função que recebe um inteiro (que pode ser positivo ou negativo) e um ponteiro para uma string. A função deve converter o inteiro para ASCII, escrevendo a string ASCII no endereço apontado pelo ponteiro.
   - Dicas
     - Para multiplicar um valor por -1, utilize neg[bwlq] (faz o complemento a 2, que é mais rápido e simples do que multiplicar)
     - O comando div só aceita um reg. ou end. de memória como divisor
   - Assuma que um inteiro ocupa 4 bytes
   - Compare sua solução com o Algoritmo (Listing) 12.12 de Plantz (2011)
   - Compare sua solução com a do professor disponibilizada no Moodle
   - Quais as principais diferenças entre as lógicas das três soluções?

Referências

- Bob Plantz. **Introduction to Computer Organization: A Guide to X86-64 Assembly Language and GNU/Linux.** 2011.
- **Intel® 64 and IA-32 Architectures Software Developer's Manual.** Intel, 2019.
- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**: **a Interface Hardware / Software.** 5a Edição. Elsevier Brasil, 2017.
- STALLINGS, W. **Arquitetura e Organização de Computadores.** 10 ed. Prentice Hall. São Paulo, 2018.
- M. Matz, J. Hubička, A. Jaeger, M. Mitchell. **System V Application Binary Interface AMD64 Architecture Processor Supplement.** 2014.