Arquitectura de Computadores I

2007/08

1º Semestre de 2007/2008

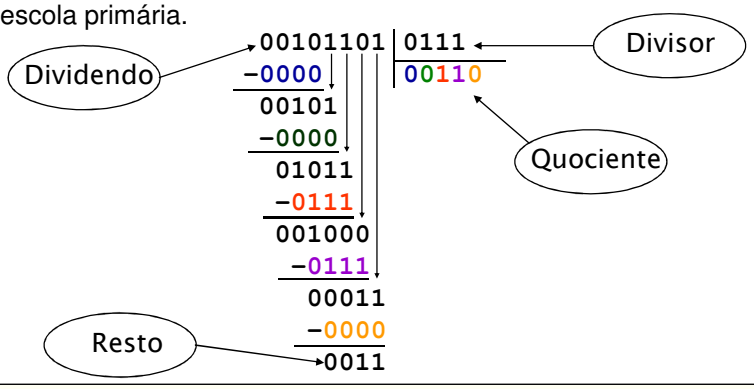
Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Arquitectura de Computadores I

2007/08

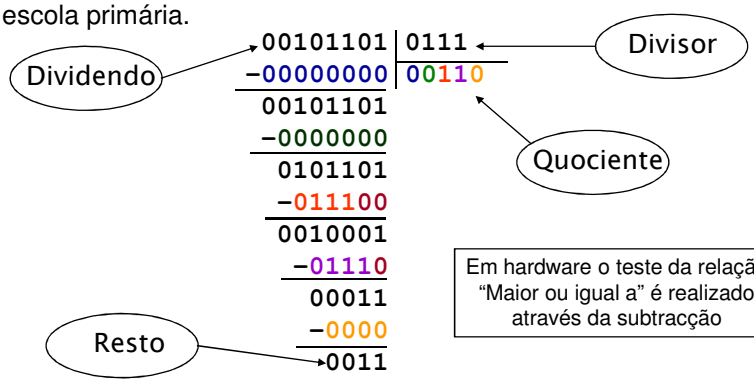
Divisão de inteiros em binário

Tal como acontecia com a multiplicação, também na divisão se usa uma arquitectura que aproveita o algoritmo que se ensina na escola primária.



Divisão de inteiros em binário

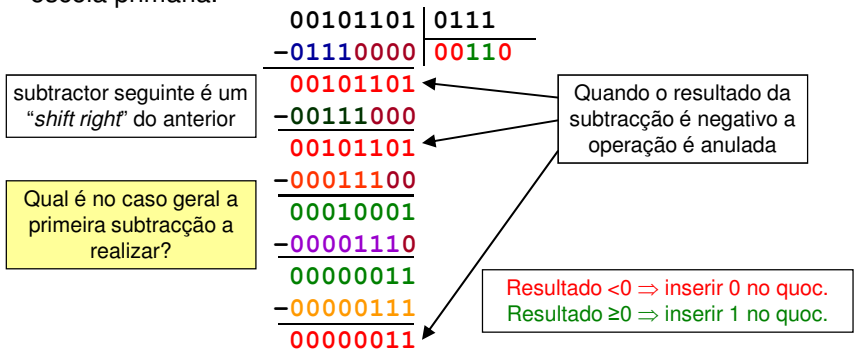
Tal como acontecia com a multiplicação, também na divisão se usa uma arquitectura que aproveita o algoritmo que se ensina na escola primária.



Em hardware o teste da relação "Maior ou igual a" é realizado através da subtracção

Divisão de inteiros em binário

Tal como acontecia com a multiplicação, também na divisão se usa uma arquitectura que aproveita o algoritmo que se ensina na escola primária.

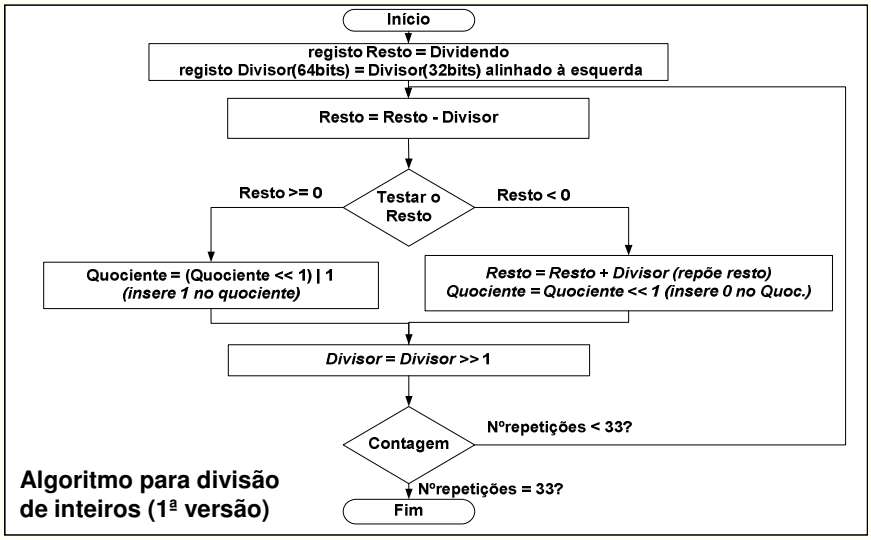


subtractor seguinte é um "*shift right*" do anterior

Quando o resultado da subtracção é negativo a operação é anulada

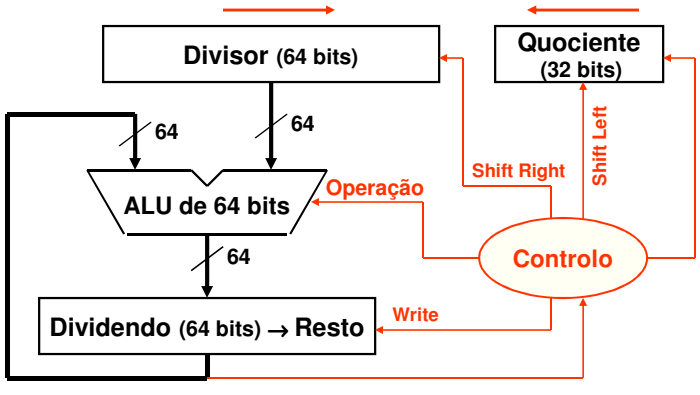
Qual é no caso geral a primeira subtracção a realizar?

Resultado <0 ⇒ inserir 0 no quoc.
Resultado ≥0 ⇒ inserir 1 no quoc.



Algoritmo para divisão de inteiros (1ª versão)

Arquitectura de um Divisor (1ª versão)

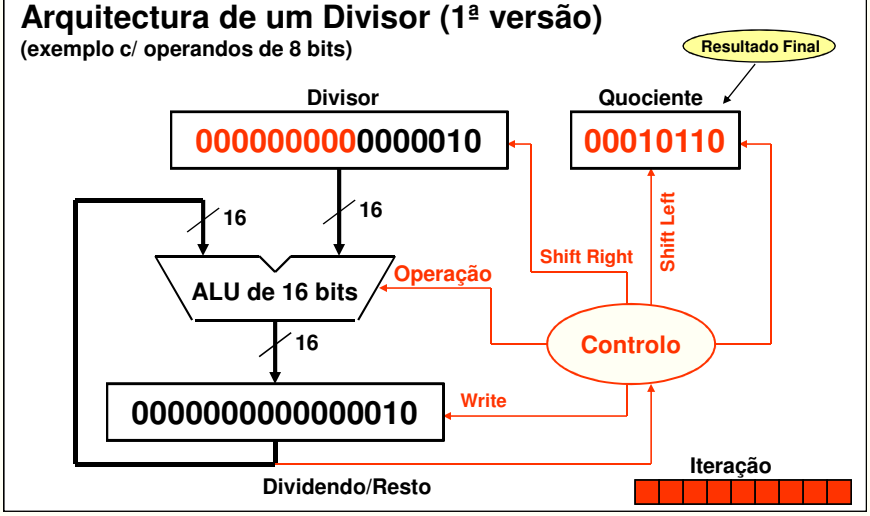


Nesta 1ª versão, a ***ALU*** e os registos ***Divisor*** e ***Dividendo/Resto*** operam com 64 bits.

A divisão de 112 (=0x70) por 5 dá um quociente de 22 (= 0x16) e um resto de 2

```
    01110000 | 0101
-0101        | 00010110
   01110000
  -101
   01110000
  -101
   01110000
  -101
   00100000
   -101
   00100000
    -101
   00001100
     -101
   00000010
      -101
   00000010
```


Arquitectura de um Divisor (1ª versão)
(exemplo c/ operandos de 8 bits)



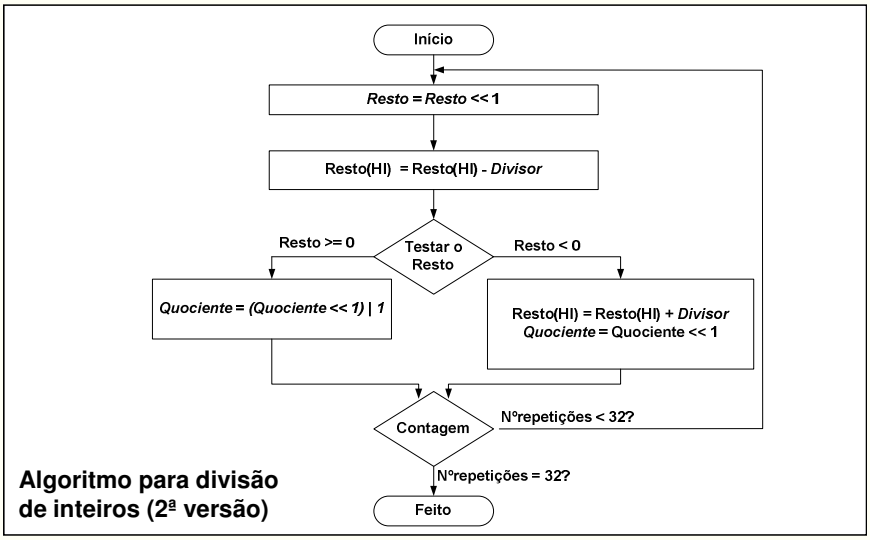
Arquitectura para divisão de inteiros (Exemplo)

Iteração	Passo	Quociente	Divisor	Resto
	Valores iniciais	0000	0010 0000	0000 0111
	Resto = Resto - Divisor	0000	0010 0000	1110 0111
1	(Resto < 0) -> +Divisor, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	shift Divisor p/ direita	0000	0001 0000	0000 0111
	Resto = Resto - Divisor	0000	0001 0000	1111 0111
2	(Resto < 0) -> +Divisor, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	shift Divisor p/ direita	0000	0000 1000	0000 0111
	Resto = Resto - Divisor	0000	0000 1000	1111 1111
3	(Resto < 0) -> +Divisor, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	shift Divisor p/ direita	0000	0000 0100	0000 0111
	Resto = Resto - Divisor	0000	0000 0100	0000 0011
4	(Resto > 0) -> sll Q, Q0 = 1	0001	0000 0100	0000 0011
	shift Divisor p/ direita	0001	0000 0010	0000 0011
	Resto = Resto - Divisor	0001	0000 0010	0000 0001
5	(Resto > 0) -> sll Q, Q0 = 1	0011	0000 0010	0000 0001
	shift Divisor p/ direita	0011	0000 0001	0000 0001

Arquitectura para divisão de inteiros

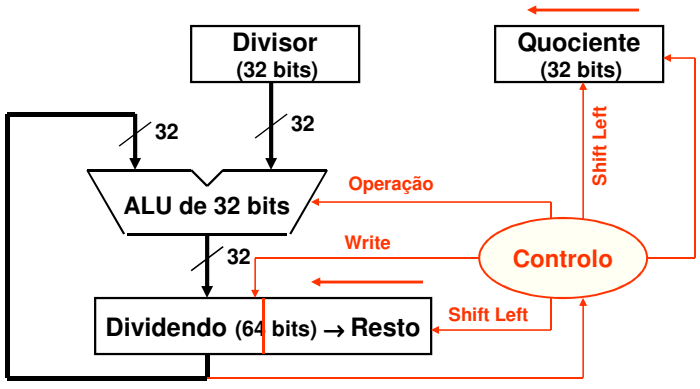
- Tal como já notáramos no caso da multiplicação, também aqui poderemos diminuir a dimensão de alguns dos registos, por se verificar que é possível manter fixo o conteúdo do registo *Divisor*, deslocando para a esquerda o conteúdo do registo *Dividendo/Resto*, sem com isso alterar o resultado final.

- Da mesma forma, também se verifica que a subtracção entre dividendo e divisor para determinação do maior dos dois pode ser feita apenas com 32 bits, permitindo assim uma redução para metade da dimensão da ALU.



Algoritmo para divisão de inteiros (2ª versão)

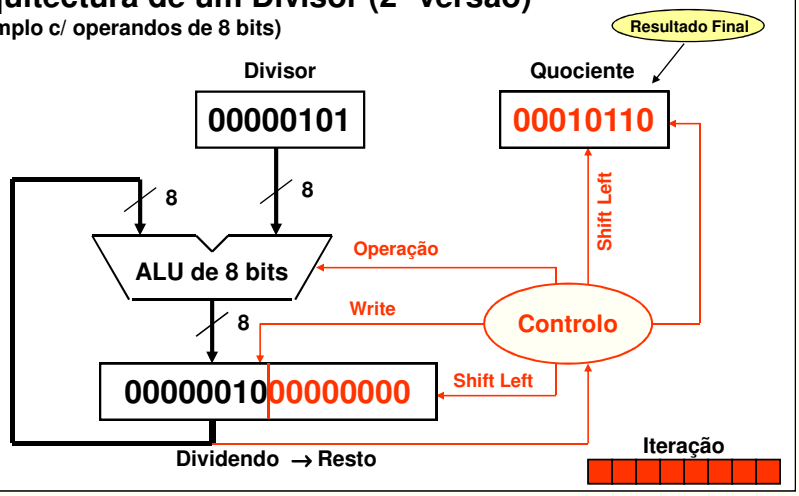
Arquitectura de um Divisor (2ª versão)



Nesta 2ª versão do divisor, a ***ALU*** e o registo ***Divisor*** operam com 32 bits.

Arquitectura de um Divisor (2ª versão)

(exemplo c/ operandos de 8 bits)



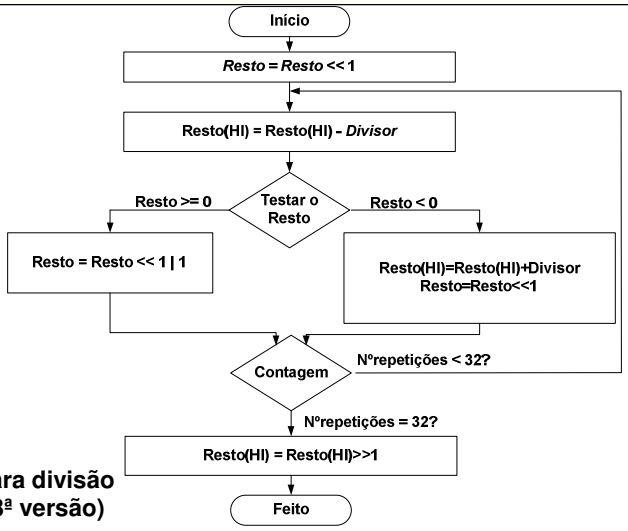
Arquitectura para divisão de inteiros (Exemplo)

Iteração	Passo	Quociente	Divisor	Resto
	Valores iniciais	0000	0010	0000 0111
	Desloca Resto p/ esquerda	0000	0010	0000 1110
1	Resto = Resto - Divisor	0000	0010	1110 1110
	(Resto < 0) -> +Divisor, sll Q, Q0 = 0	0000	0010	0000 1110
	Desloca Resto p/ esquerda	0000	0010	0001 1100
2	Resto = Resto - Divisor	0000	0010	1111 1100
	(Resto < 0) -> +Divisor, sll Q, Q0 = 0	0000	0010	0001 1100
	Desloca Resto p/ esquerda	0000	0010	0011 1000
3	Resto = Resto - Divisor	0000	0010	0001 1000
	(Resto > 0) -> sll Q, Q0 = 1	0001	0010	0001 1000
	Desloca Resto p/ esquerda	0001	0010	0011 0000
4	Resto = Resto - Divisor	0001	0010	0001 0000
	(Resto > 0) -> sll Q, Q0 = 1	0011	0010	0001 0000

Arquitectura para divisão de inteiros

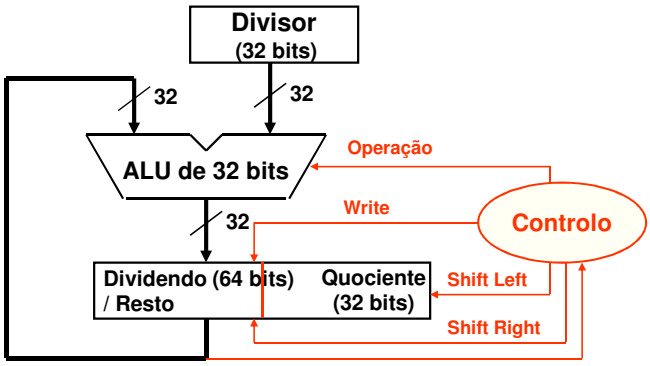
- Finalmente, pode verificar-se que o deslocamento à esquerda do conteúdo do registo *Dividendo*, é acompanhado por um deslocamento idêntico do registo *Quociente*.

- Uma vez que por cada deslocamento à esquerda do *Dividendo*, seria acrescentado um "0" à sua direita, não há incompatibilidade pelo facto de substituir esse "0" pelo novo dígito do *Quociente*. Dessa forma poupa-se ainda o espaço que seria necessário para armazenar esse quociente.



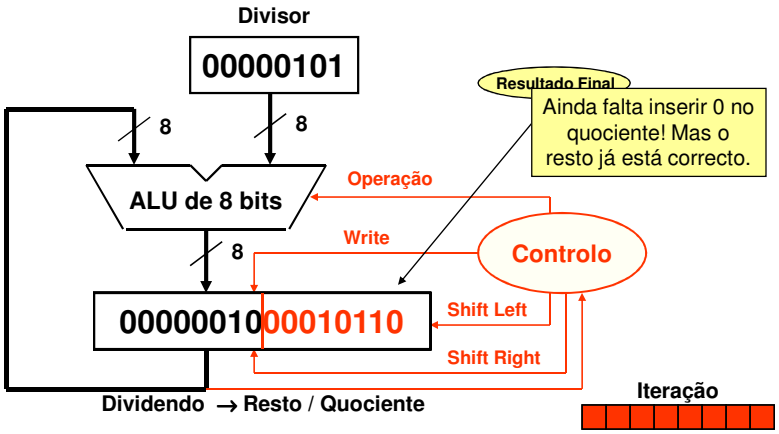
Algoritmo para divisão de inteiros (3ª versão)

Arquitectura de um Divisor (versão final)



Na versão final do divisor, o registo ***Quociente*** desaparece, sendo substituído pela parte menos significativa do registo ***Dividendo/Resto***.

Arquitectura de um Divisor (versão final)
(exemplo c/ operandos de 8 bits)



Arquitectura para divisão de inteiros (Exemplo)

Iteração	Passo	Divisor	Resto
0	Valores iniciais	0010	0000 0111
	Desloca Resto p/ esquerda	0010	0000 1110
1	Resto = Resto - Divisor	0010	1110 1110
	(Resto < 0) -> +Divisor, sll R, R0 = 0	0010	0001 1100
2	Resto = Resto - Divisor	0010	1111 1100
	(Resto < 0) -> +Divisor, sll R, R0 = 0	0010	0011 1000
3	Resto = Resto - Divisor	0010	0001 1000
	(Resto > 0) -> sll R, R0 = 1	0010	0011 0001
4	Resto = Resto - Divisor	0010	0001 0001
	(Resto > 0) -> sll R, R0 = 1	0010	0010 0011
	Desloca Resto p/ direita	0010	0001 0011

Divisão de inteiros com sinal

A divisão de inteiros com sinal faz-se em sinal e módulo

Nas divisões com sinal aplicam-se as seguintes regras:

- Dividem-se dividendo por divisor em módulo
- O quociente terá sinal negativo se os sinais de dividendo e divisor forem diferentes
- O resto terá o mesmo sinal que o dividendo

Exemplos: -7 / 3 = -2 c/ resto = -1 7 / -3 = -2 c/ resto = 1

A Divisão de inteiros no MIPS

- No MIPS, a divisão é assegurada por uma arquitectura semelhante à anteriormente descrita para a multiplicação. Tal como acontecia naquele caso, continua a existir a necessidade de um registo de 64 bits para armazenar o valor inicial do dividendo, e bem assim o resultado final na forma de um quociente e de um resto.

- Os mesmos registos, **HI** e **LO**, que tinham já sido apresentados para o caso da multiplicação, são igualmente utilizados para a divisão:
  - o registo **HI** armazena o **resto da divisão** inteira
  - o registo **LO** armazena o **quociente da divisão** inteira

A Divisão de inteiros no MIPS

Em *Assembly*, a divisão é efectuada pela instrução

```
div   $reg1, $reg2      # Divide
divu  $reg1, $reg2      # Divide unsigned
```

em que $reg1 é o dividendo e $reg2 o divisor. O resultado fica armazenado nos registos HI (resto) e LO (quociente).

A transferência de informação entre os registos HI e LO e os restantes registos de uso geral faz-se através das instruções:

```
mfhi  $reg      # move from hi - Copia HI para $reg
mflo  $reg      # move from lo - Copia LO para $reg
mthi  $reg      # move to hi - Copia $reg para HI
mtlo  $reg      # move to lo - Copia $reg para LO
```