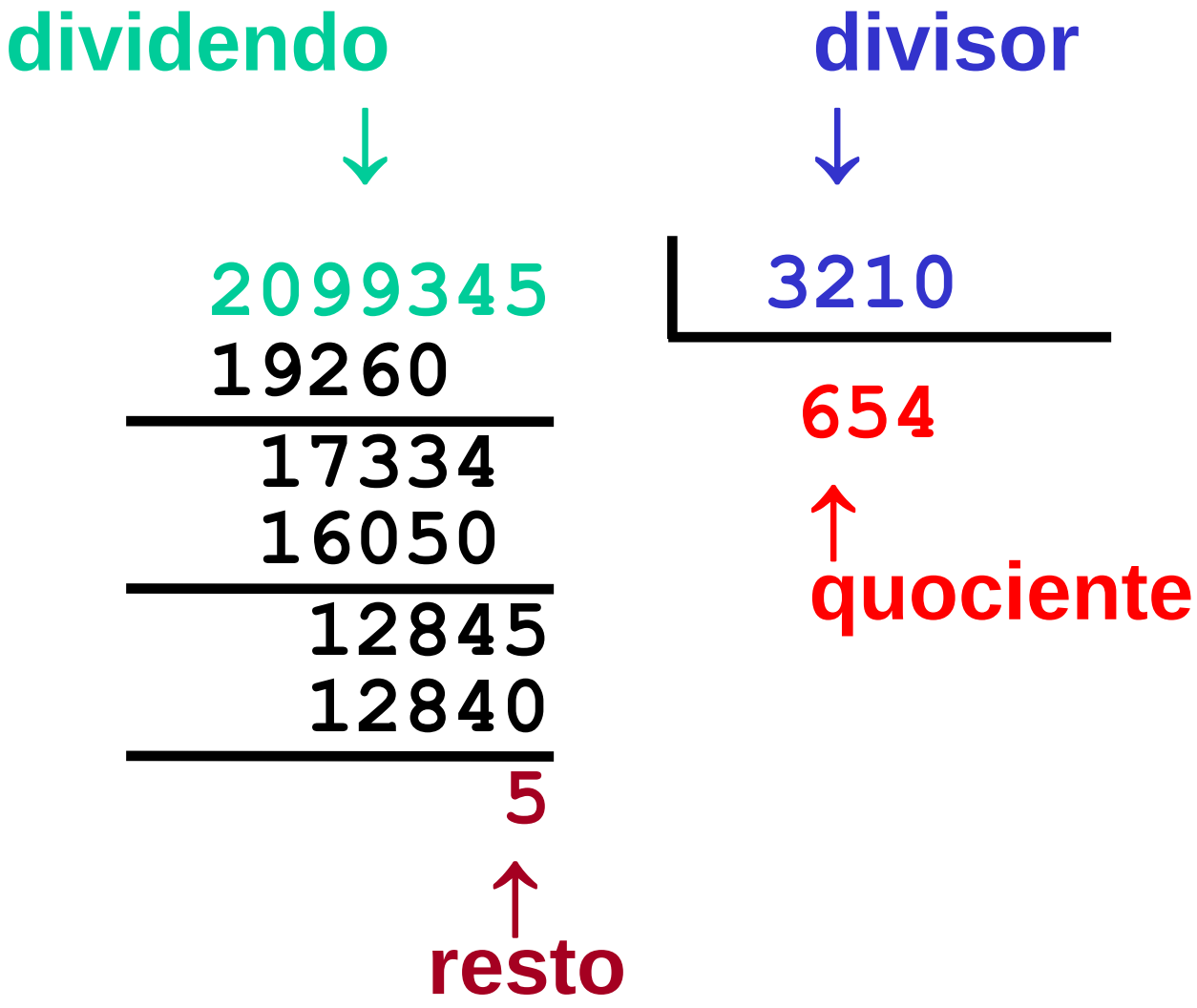
Divisão em binário

Outra aplicação para o Ahmes !!!

Divisão no sistema decimal (números inteiros positivos)



Divisão no sistema decimal (números inteiros positivos)

```
2099345 | 3210
        |
        |
        ????
```

- Operação inversa da multiplicação
- Quociente gerado da esquerda para a direita
- Subtrações sucessivas (dividendo - divisor)
- Resto só pode ser zero ou positivo (se divisor maior do que dividendo, o quociente é zero e o resto é o próprio dividendo)
- Primeira subtração: por “tentativa e erro” - somente pode ser feita a subtração se o resultado não ficar negativo nem maior que o divisor

Divisão no sistema decimal (números inteiros positivos)

Tentativa 1: resultado negativo = erro

Divisão no sistema decimal (números inteiros positivos)

```
  2099345 | 3210
-3210     |------
-3190     | 1 ?
```

Tentativa 2: resultado negativo = erro

Divisão no sistema decimal (números inteiros positivos)

```
  2099345 | 3210
-3210     |------
----------| 1 ?
 -3001
```

Tentativa 3: resultado negativo = erro

Divisão no sistema decimal (números inteiros positivos)

```
  2099345 | 3210
-3210     |------
---------- | 1 ?
 -1111
```

Tentativa 4: resultado negativo = erro

Divisão no sistema decimal (números inteiros positivos)

```
  2099345 | 3210
 -3210    |------
 -------  | 1 ?
  17783
```

Tentativa 5:
resultado maior que divisor = erro

Divisão no sistema decimal (números inteiros positivos)

```
 2099345 | 3210
-6420    |------
---------| 2 ?
 14573
```

Tentativa 6:
resultado maior que divisor = erro

Divisão no sistema decimal (números inteiros positivos)

```
  2099345 | 3210
-19260    |------
----------| 6 !!!
    1733
```

Tentativa 10: resultado positivo e menor do que o quociente - certo !!!

Divisão no sistema decimal (números inteiros positivos)

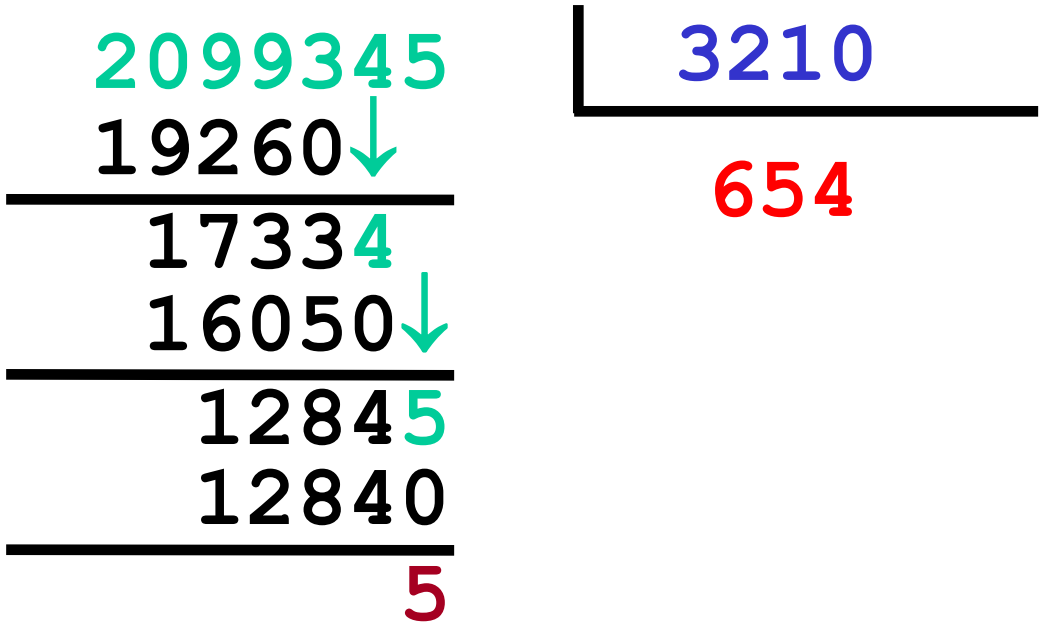
```
  2099345 | 3210
 -19260   |------
 -------  | 6 !!!
   1733
```

Na prática, fazemos isto em 2 etapas:

1. determinar quantos dígitos do dividendo são necessários para obter valor ≥ divisor

2. determinar quantas vezes o divisor "cabe" neste valor

Divisão no sistema decimal (números inteiros positivos)



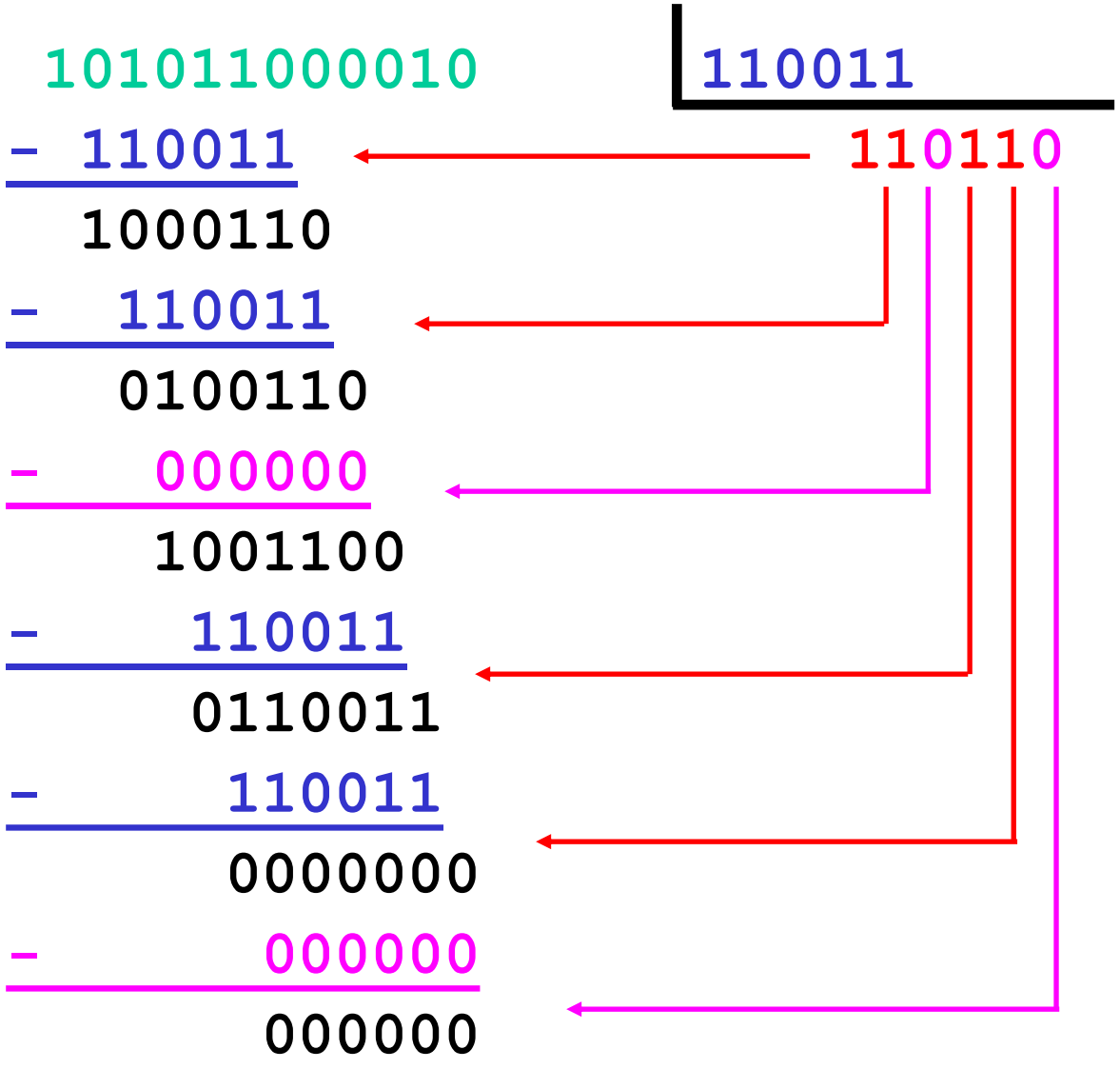
Divisão no sistema decimal
(números inteiros positivos)

```
                          2099345  | 3210
6x3210x100   =  1926000   |______________
                -------      654
                 173345
5x3210x 10   =   160500
                -------
                  12845
4x3210x  1   =    12840
                -------
                      5
```

7 dígitos ÷ 4 dígitos = 3 dígitos

(e o resto pode ter até 4 dígitos !)

Divisão no sistema binário - inteiros positivos



Divisão no sistema binário - inteiros positivos

```
 101011000010   | 110011
-  110011       |
   100011       |      1
```

- Para iniciar a divisão, precisamos determinar quantos dígitos são necessários para obter valor maior ou igual ao divisor
- O primeiro dígito do quociente só pode ser **1** (se conseguimos um valor maior do que o divisor) ou **0** (não é possível dividir)

Divisão no sistema binário - inteiros positivos

```
 101011000010 | 110011
-  110011     |------
   1000110    |   11
-   110011
    010011
```

- A partir do segundo dígito, “baixamos” um dígito do dividendo e tentamos nova subtração do divisor.
- O próximo dígito do quociente pode ser **1** (se conseguimos subtrair) ou **0** (não é possível subtrair)

Divisão no sistema binário - inteiros positivos

```
 101011000010 | 110011
-  110011     |------
   1000110    |   110
-   110011    |
    0100110
-    000000
     100110
```

- A partir do segundo dígito, “baixamos” um dígito do dividendo e tentamos nova subtração do divisor.
- O próximo dígito do quociente pode ser **1** (se conseguimos subtrair) ou **0** (não é possível subtrair)

Divisão no sistema binário - inteiros positivos

```
 101011000010 | 110011
-  110011     |------
   1000110    |  1101
-   110011
   0100110
-    000000
   1001100
-    110011
   011001
```

- A partir do segundo dígito, “baixamos” um dígito do dividendo e tentamos nova subtração do divisor.
- O próximo dígito do quociente pode ser **1** (se conseguimos subtrair) ou **0** (não é possível subtrair)

Divisão no sistema binário - inteiros positivos

```
 101011000010 | 110011
-  110011     |-------
   1000110    |  11011
-   110011    |
    0100110   |
-    000000   |
     1001100  |
-     110011  |
      0110011 |
-      110011 |
       000000 |
```

- A partir do segundo dígito, “baixamos” um dígito do dividendo e tentamos nova subtração do divisor.
- O próximo dígito do quociente pode ser **1** (se conseguimos subtrair) ou **0** (não é possível subtrair)

Divisão no sistema binário - inteiros positivos

```
 101011000010 | 110011
-  110011     |------
   1000110    |  110110
-   110011
   0100110
-    000000
    1001100
-    110011
    0110011
-    110011
    0000000
-   000000
    000000
```

- A partir do segundo dígito, “baixamos” um dígito do dividendo e tentamos nova subtração do divisor.
- O próximo dígito do quociente pode ser **1** (se conseguimos subtrair) ou **0** (não é possível subtrair)

Divisão no sistema binário - inteiros positivos

```
 1010110 000010   | 110011
-  110011         |------
   1000110        | 110110
-   110011
    0100110
-    000000
     1001100
-     110011
      0110011
-      110011
       0000000
-       000000
        000000
```

- dividendo com m dígitos
- **divisor** com **n** dígitos
- quociente até m-n dígitos
- resto até n dígitos

- Nos algoritmos para computadores:
- dividendo tem 2n bits e os demais n bits
- pode ocorrer estouro
- divisão por zero é erro

Algoritmo básico (dividendo com 2n bits e divisor com n bits)

1. Início: resto $\leftarrow$ (n+1 msbits do dividendo), i $\leftarrow$ n
2. Se resto $\geq$ divisor, então resto $\leftarrow$ resto - divisor e lsbit do quociente = 1, senão lsbit do quociente = 0
3. i $\leftarrow$ i - 1; se i = 0, terminar (resto e quociente estão corretos)
4. Deslocar resto para a esquerda em 1 bit, mantendo o msbit (r com n+1 bits) e colocando como novo lsbit de resto o próximo bit do dividendo original. Deslocar quociente para a esquerda; voltar ao passo 2.

Divisão no sistema binário - inteiros positivos

```
 10101100010 | 110011
-  110011    |------
   1000110   | 110110
-   110011   |
    0100110  |
-    000000  |
     1001100 |
-     110011 |
      0110011|
-      110011|
       0000000|
-       000000|
        000000|
```

Problema que pode ocorrer na primeira subtração neste algoritmo:

"resto" - divisor ≥ divisor

Neste caso, a divisão provocaria estouro, e não pode ser realizada.

```
 1101000000010 | 110011
-  110011      |------
   110101      | 10
```

Algoritmo com testes: estouro e divisão por 0 (Dd=dividendo, r=resto, v=divisor, q=quociente)

1. Início: se v = 0, terminar: divisão por zero;

   se D≥v (D-v não provoca “borrow”), terminar: estouro.
   Senão, a divisão pode ser feita; neste caso,
   fazer i ← n, r ← (n+1 msbits do dividendo).

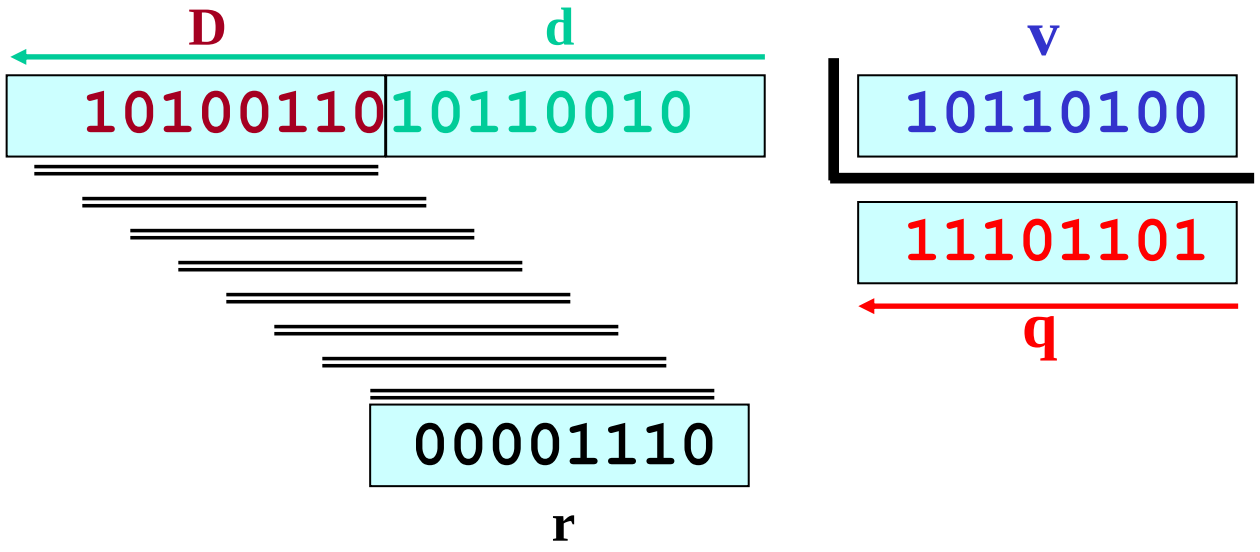
2. Se r ≥ v, então r ← r - v e lsbit de q = 1,

   senão lsbit de q = 0

3. i ← i - 1; se i = 0, terminar (resto em r e quociente em q estão corretos)

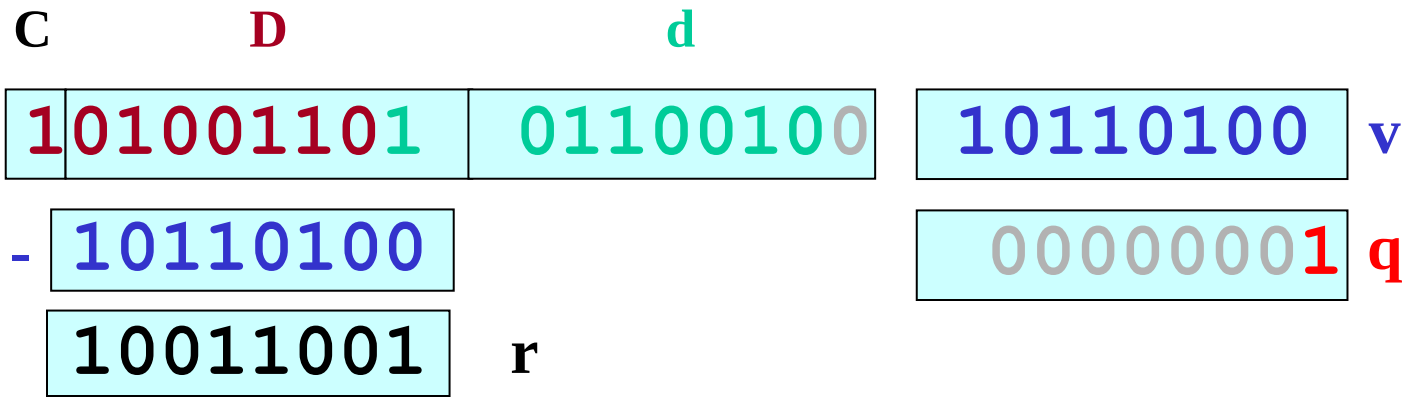
4. Deslocar r para a esquerda em 1 bit, mantendo o msbit (r com n+1 bits) e colocando como novo lsbit de resto o próximo bit do dividendo original. Deslocar q para a esquerda; voltar ao passo 2.

Preparando o algoritmo para o Ahmes: dividendo com 16 bits, demais valores com 8 bits (Dd=dividendo, r=resto, v=divisor, q=quociente)



- o quociente é formado da esquerda para a direita, iniciando no bit 0 de q e deslocando para a esquerda a cada iteração
- o dividendo (Dd) também é deslocado para a esquerda a cada iteração, para que a operação r ← r - v possa ser substituída por D ← D - v

Preparando o algoritmo para o Ahmes: dividendo com 16 bits, demais valores com 8 bits (Dd=dividendo, r=resto, v=divisor, q=quociente)



- o valor “r” usado no algoritmo básico, na verdade será formado pelo carry (C) seguido de D
- se C = 1, certamente r > v
- se C = 0, calculando D - v se obtém o mesmo valor que “r - v” e, neste caso, basta testar se D ≥ v (ou seja, se não ocorre “borrow” quando se calcula D - v)
- *nota*: o “r” obtido na subtração substitui o valor de “D” para a próxima iteração

Preparando o algoritmo para o Ahmes:
(Dd=dividendo, r=resto, v=divisor, q=quociente)

1. Início: se v = 0, terminar: divisão por zero;

   se D≥v (D-v não provoca “borrow”), terminar: estouro.

2. i ← n, q ← 0.

3. Deslocar q para a esquerda em 1 bit, abrindo espaço para o novo bit que será calculado nesta iteração. Deslocar D e d para a esquerda (carry ← msbit de D).

   Se carry = 1, D > v. Se carry = 0 e D-v não provoca borrow, então D ≥ v. Nos dois casos, fazer D ← D - v e colocar 1 no lsbit de q.

   Senão, deixar D inalterado e deixar o 0 no lsbit de q.

4. i ← i - 1. Se i = 0, terminar (resto em r e quociente em q estão corretos); senão, voltar ao passo 3.

Algoritmo para o Ahmes (código simbólico p/Daedalus)

```
; definição de variáveis e constantes
        ORG 128   ; começando na palavra 128
Dms:    DB 0      ; msbits do dividendo
dls:    DB 0      ; lsbits do dividendo
v:      DB 0      ; divisor
q:      DB 0      ; quociente
r:      DB 0      ; resto
est:    DB 0      ; estado: estouro = -1
                  ;         div. por zero = 0
                  ;         normal = 1
i:      DB 0      ; contador
zero:   DB 0      ; constante 0
um:     DB 1      ; constante 1
mum:    DB 255    ; constante -1
oito:   DB 8      ; constante 8

Dmst:   DB 0      ; msbits do dividendo - temp
dlst:   DB 0      ; lsbits do dividendo - temp
```

Algoritmo para o Ahmes (código simbólico p/Daedalus)

```
; teste de exceções: estouro e div. por zero
inicio:
      LDA v
      JZ  div_por_zero ; divisor = 0
      LDA Dms
      SUB v
      JNB estouro ; D >= v provoca estouro
; não ocorreu nenhuma exceção: inicialização
      LDA Dms
      STA Dmst ; Dms temporário
      LDA dls
      STA dlst ; dls temporário
      LDA oito
      STA i    ; contador = 8
      LDA zero
      STA q    ; quociente = 0
```

Algoritmo para o Ahmes (código simbólico p/Daedalus)

```
repetir:              ; início do laço
      LDA q
      SHL
      STA q           ; desloca q p/esquerda
      LDA dlst
      SHL
      STA dlst        ; desloca d p/esquerda
      LDA Dmst
      ROL             ; desloca D p/esquerda
      STA Dmst        ; msbit de D em carry
      JC  Dms_maior   ; se C=1, D > v
      LDA Dmst
      SUB v           ; calcula D - v
      JB  Dms_menor   ; se B=1, D < v
```

Algoritmo para o Ahmes (código simbólico p/Daedalus)

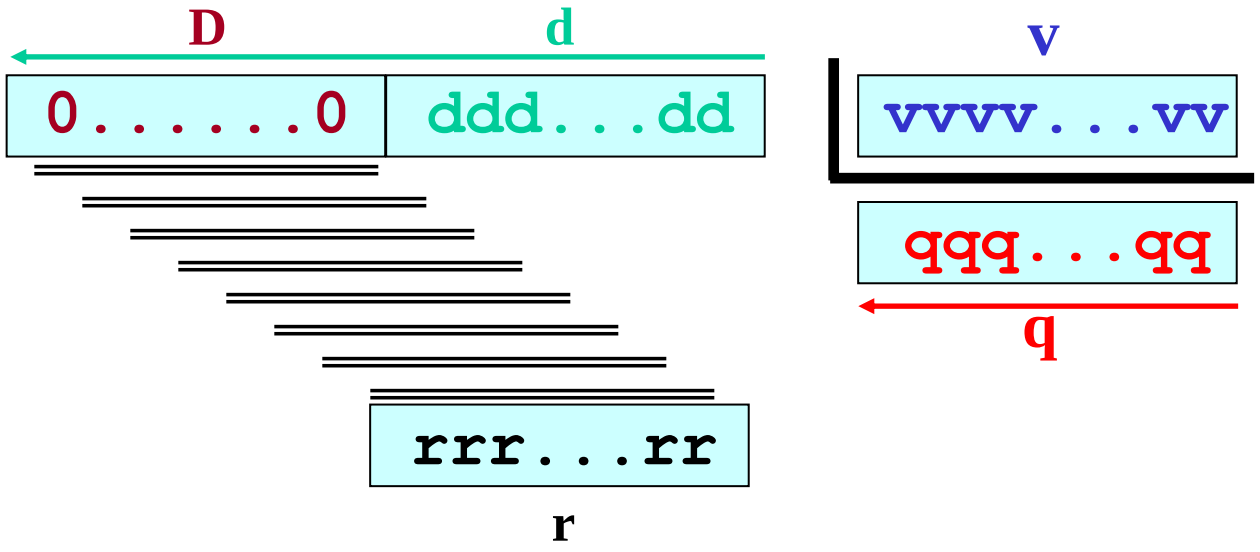
```
Dms_maior:
      LDA Dmst      ; se D >= v
      SUB v         ; pode subtrair
      STA Dmst
      LDA q         ; e novo dígito de q
      OR  um        ; deve ser igual a 1
      STA q
Dms_menor:
      LDA i
      SUB um        ; decrementa contador
      STA i
      JNZ repetir   ; se i > 0, repete o laço
```

Algoritmo para o Ahmes (código simbólico p/Daedalus)

```
fim:                ; fim normal
      LDA Dmst
      STA r         ; resto = Dmst
      LDA um        ; estado = 1
      JMP estado
div_por_zero:       ; fim com erro: div. por zero
      LDA zero      ; estado = 0
      JMP estado
estouro:            ; fim com erro: estouro
      LDA mum       ; estado = -1
estado:
      STA est       ; armazena estado
      HLT
```

Preparando os operandos: valores de 8 bits

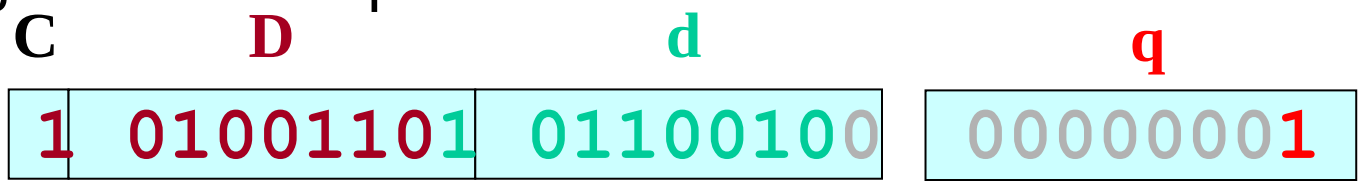
(Dd=dividendo, r=resto, v=divisor, q=quociente)



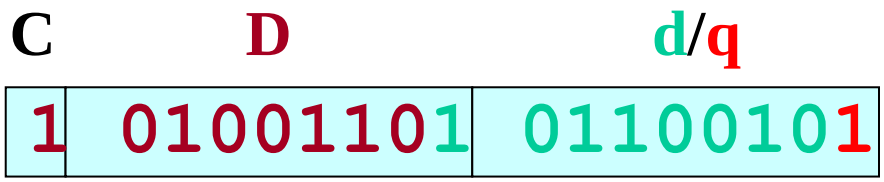
- O dividendo (em 8 bits) deve ser expandido para 16 bits
- Como os operandos são inteiros positivos, então:
  D = 000000000
  d = dividendo (8 bits)
- Demais operandos todos em 8 bits (divisor, quociente, resto)

O que pode ser melhorado (1) ?

- os bits de **d** (dls) são deslocados para a esquerda, passando para **D** (Dms) e liberando os bits menos significativos de **d**
- os bits de **q** são construídos a partir do bit menos significativo da palavra

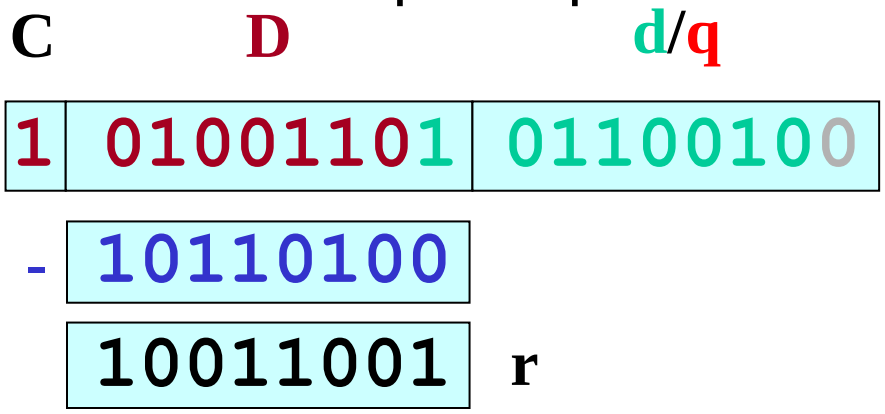


- conclusão: pode ser usada uma mesma palavra para guardar **q** e **d**

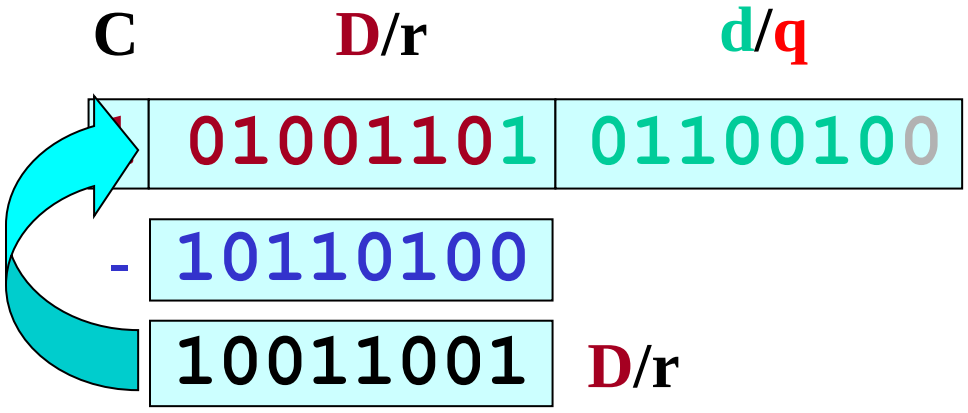


O que pode ser melhorado (2) ?

- a cada iteração, o divisor é subtraído da variável **D** (Dms) e no final o valor desta é copiado para **r**



- conclusão: também pode ser usada uma mesma palavra para guardar **D** e **r**



Algoritmo com melhorias 1 e 2
(Dd=dividendo, r=resto, v=divisor, q=quociente)

1. Início: se v = 0, terminar: divisão por zero;

   se D≥v (D-v não provoca “borrow”), terminar: estouro.

2. i ← n , q ← d , **r** ← D.

3. Deslocar **r** e q (D e d) para a esquerda (carry ← msbit de **r**/D). Se carry = 1, D > v. Se carry = 0 e D-v não provoca borrow, então D ≥ v. Nos dois casos, fazer D ← D - v e colocar 1 no lsbit de d (q).

   Senão, deixar **r** e q (D e d) inalterados (0 no lsbit de q).

4. i ← i - 1. Se i = 0, terminar (resto em r/D e quociente em q/d estão corretos); senão, voltar ao passo 3.

Programa para o Ahmes

```
; definição de variáveis
      ORG 128        ; começando na palavra 128
Dr:   DB 0           ; msbits do dividendo
dq:   DB 0           ; lsbits do dividendo
v:    DB 0           ; divisor
dqt:  DB 0           ; lsbits do dividendo (no final = quociente)
Drt:  DB 0           ; msbits do dividendo (no final, = resto
est:  DB 0           ; estado: estouro = -1
                     ;         div. por zero = 0
                     ;         normal = 1
i:    DB 0           ; contador

; definição de constantes
zero: DB 0           ; constante 0
um:   DB 1           ; constante 1
mum:  DB -1          ; constante -1
oito: DB 8           ; constante 8
```

Programa para o Ahmes

```
; teste de exceções: estouro e div. por zero
inicio:
     LDA v
     JZ  div_por_zero ; divisor = 0
     LDA Dr
     SUB v
     JNB estouro   ; D >= v provoca estouro
; não ocorreu nenhuma exceção: inicialização
     LDA Dr
     STA Drt       ; Dr temporário
     LDA dq
     STA dqt       ; dq temporário
     LDA oito
     STA i         ; contador = 8
repetir:           ; início do laço
     LDA dqt
     SHL
     STA dqt       ; desloca dq p/esquerda
     LDA Drt
     ROL           ; desloca Dr p/esquerda
     STA Drt       ; msbit de Dr em carry
     JC  Dr_maior  ; se C=1, D > v
```

Programa para o Ahmes

```
      LDA Drt
      SUB v          ; calcula D - v
      JB  Dr_menor   ; se B=1, D < v

Dr_maior:
      LDA Drt        ; se D >= v
      SUB v          ; pode subtrair
      STA Drt
      LDA dqt        ; e novo dígito de dq
      OR  um         ; deve ser igual a 1
      STA dqt

Dr_menor:
      LDA i
      SUB um         ; decrementa contador
      STA i
      JNZ repetir    ; se i > 0, repete o laço
```

Programa para o Ahmes

```
fim:                ; fim normal
     LDA um         ; estado = 1

estado:
     STA est        ; armazena estado
     HLT

div_por_zero:       ; fim com erro: div. por zero
     LDA zero       ; estado = 0
     JMP estado

estouro:            ; fim com erro: estouro
     LDA mum        ; estado = -1
     JMP estado
```

Divisão de números positivos em complemento de 2

- quando valores positivos estão representados em complemento de 2, podemos testar se o resultado da subtração (D - v) é negativo para determinar se é possível subtrair ou não (notar que, como o bit de sinal de D é sempre 0, não é mais necessário testar o carry para saber se D>v)
- caso o resultado da subtração D-v seja negativo, somar v ao resultado obtido restaura o valor de D
- o método assim modificado, é chamado de "divisão com restauração"

Algoritmo de divisão "com restauração" (valores positivos em complemento de 2)

1. Início: se v = 0, terminar: divisão por zero.

   Senão, deslocar Dd para a esquerda (msbit sempre = 0); se D≥v (D-v não provoca "borrow"), terminar: estouro.
   Senão, fazer: i ← n , q ← d , **r** ← D.

2. Fazer **r** ← **r** - v. Se **r** positivo, colocar 1 no lsbit de d (q).
   Senão, colocar 0 no lsbit de d (q) e "restaurar **r**" fazendo **r** ← **r** + v (na prática, basta não salvar **r**).

3. Fazer i ← i - 1. Se i = 0, terminar (resto em **r**/D e quociente em q/d estão corretos).

4. Deslocar Dd (**r** q) para a esquerda, colocando 0 no lsbit de q. Voltar para o passo 2.

Programa para o Ahmes

```
; ************************************************
; * Algoritmo para divisão de inteiros em complemento de 2 *
; * Equivale ao programa da página 78 do livro - 2a edição *
; ************************************************
; teste de exceções: estouro e div. por zero
inicio:
     LDA v
     JZ  div_por_zero ; divisor = 0
     LDA dq
     SHL              ; desloca dq p/esquerda
     STA dqt          ; e salva em dq temporário
     LDA Dr           ; desloca Dr p/esquerda
     ROL              ; msbit de Dr em carry
     STA Drt          ; e salva em Dr temporário
     SUB v            ; calcula D - v
     JP estouro       ; Se positivo (D >= v), estouro
; não ocorreu nenhuma exceção: inicializa contador
; atenção: Drt e dqt já estão inicializados !!!
     LDA oito
     STA i            ; contador = 8
```

Programa para o Ahmes

```
repetir:            ; início do laço
     LDA Drt
     SUB v          ; calcula D - v
     JN  nao_salva  ; se negativo, D < v
     STA Drt        ; se positivo, atualizar r
     LDA dqt
     OR  um         ; faz bit do quociente = 1
     STA dqt
nao_salva:          ; eqüivale a "restaurar r"
     LDA i
     SUB um         ; decrementa contador
     STA i
     JZ  fim        ; se i = 0, fim normal
     LDA dqt
     SHL            ; desloca dq p/esquerda
     STA dqt
     LDA Drt
     ROL            ; desloca Dr p/esquerda
     STA Drt
     JMP repetir    ; volta para o início do laço
```

Programa para o Ahmes

```
fim:                ; fim normal
     LDA um         ; estado = 1
estado:
     STA est        ; armazena estado
     HLT
div_por_zero:       ; fim com erro: div. por zero
     LDA zero       ; estado = 0
     JMP estado
estouro:            ; fim com erro: estouro
     LDA mum        ; estado = -1
     JMP estado

; definição de variáveis
      ORG 128       ; começando na palavra 128
Dr:   DB 0          ; msbits do dividendo
dq:   DB 0          ; lsbits do dividendo
v:    DB 0          ; divisor
dqt:  DB 0          ; lsbits do dividendo (no final = quociente)
Drt:  DB 0          ; msbits do dividendo (no final, = resto)
est:  DB 0          ; estado: estouro = -1
                    ;         div. por zero = 0
                    ;         normal = 1
i:    DB 0          ; contador

; definição de constantes
zero: DB 0          ; constante 0
um:   DB 1          ; constante 1
mum:  DB -1         ; constante -1
oito: DB 8          ; constante 8
```

Divisão de números positivos em complemento de 2 sem restauração

- no algoritmo “com restauração”, quando é preciso fazer uma restauração (somar v a r_i) durante uma iteração i, é gerado um bit 0 no quociente e, na iteração seguinte (i-1) certamente v precisa ser subtraído de r_{i-1} para determinar se o dígito seguinte do quociente será 0 ou 1
- como r, na iteração i-1, está deslocado para a esquerda em um bit, ele vale $2r_i$; logo, na iteração i-1 teremos:

  $r_{i-1} \leftarrow 2(r_i + v) - v = 2r_i + 2v - v = 2r_i + v$
- logo, após obter um bit 0 no quociente, em vez de fazer a restauração de r naquela iteração, basta somar v a r na próxima iteração em vez de fazer a subtração
- com isso, a cada iteração, teremos apenas uma soma ***ou*** uma subtração (em vez de 3n/2 operações, em média, no algoritmo com restauração)

Divisão de números positivos ou negativos em complemento de 2

- devem ser tomados cuidados especiais em relação aos seguintes aspectos:
  - determinação de estouro
  - acerto dos sinais do quociente e do resto
- as diretrizes para implementação de um algoritmo para este tipo de divisão podem ser encontradas no Seção 6.5 do livro texto
- este assunto é deixado como estudo opcional para aqueles que desejarem se “especializar” em divisão binária