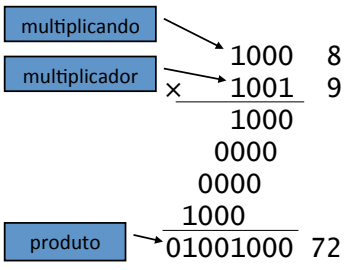
Arquitetura de Computadores I
Multiplicação e Divisão binárias

António de Brito Ferrari
ferrari@ua.pt

Multiplicação

Unsigned



Operandos: 0..(2^n-1) Produto: 0..(2^{2n}-2^{n+1}+1)
No. de bits do produto = 2n-bits (soma do no. de bits dos operandos)

Signed

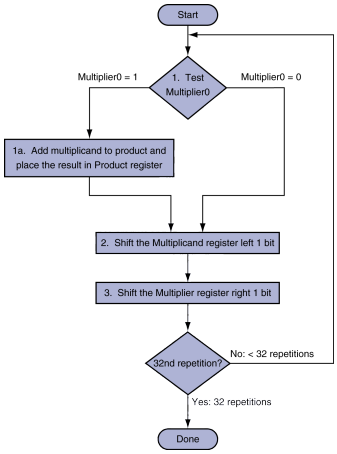
$X*Y = X*y_0 + X*2y_1 + \ldots + X*2^{(n-2)}y_{n-2} - X*2^{(n-1)}y_{n-1}$

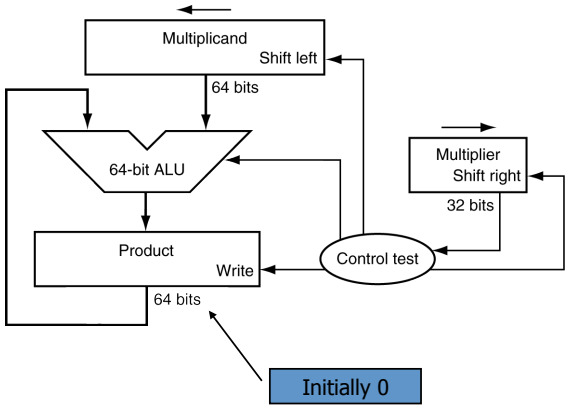
```
    1000   -8 = X
  × 1001   -7 = Y
11111000  ← sign-extension
0000000
000000        2's complement
01000   ←     do multiplicando
00111000  +56
```

Operandos: $-2^{n-1}..+(2^{n-1}-1)$
Produto: $(-2^{2n-2}+2^{n+1})..+2^{2n-2}$ ← 2n-bits

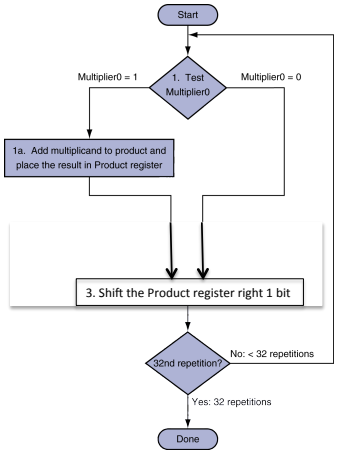
Multiplicador série

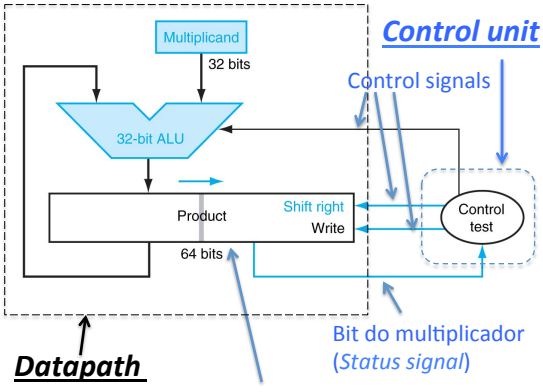
Algoritmo Add-shift



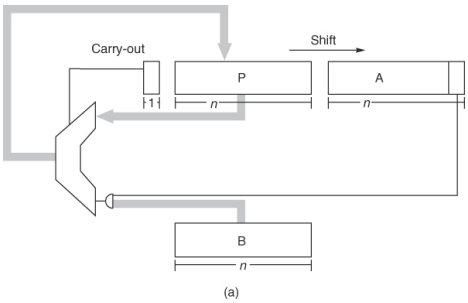


Multiplicador Série otimizado



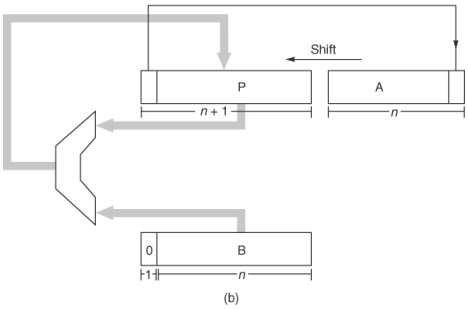


Multiplicador inicialmente na metade direita
Produto parcial na metade esquerda



Multiplicador

A – Multiplicador
B – Multiplicando
P_A – Resultado Parcial

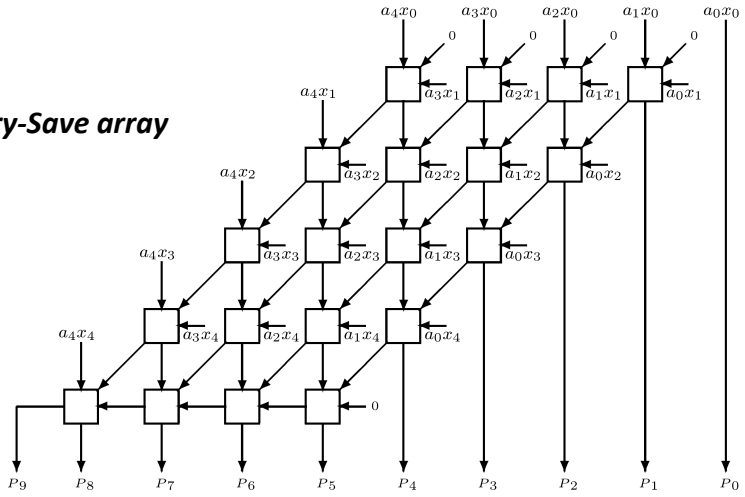


Divisor

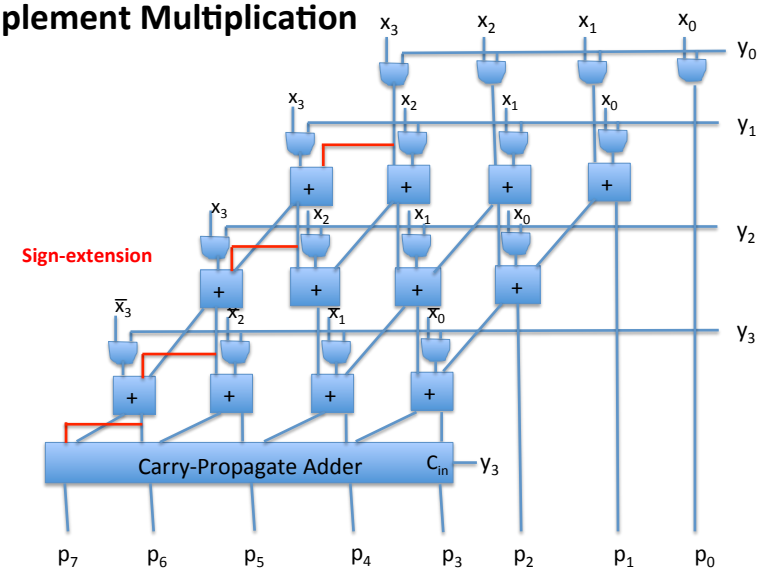
A – Dividendo_Quociente
B – Divisor
P – Resto

Multiplicador Paralelo
Array Multiplier (unsigned)

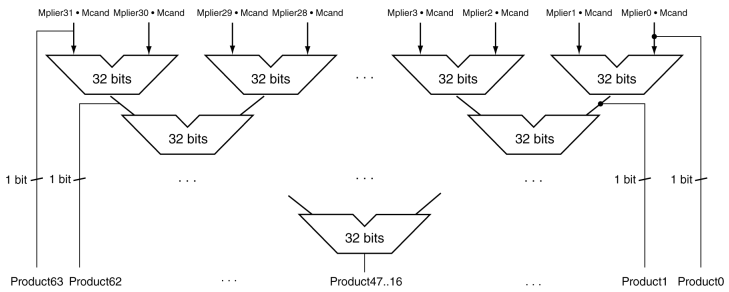
Carry-Save array



Carry-Save Array
2's complement Multiplication



Multiplicador Paralelo em árvore: Wallace tree



Algoritmo de Booth (signed multiplication)

Recodificação do multiplicador:

y_i	y_{i-1}	Operação	
0	0	Shift	
0	1	Add - Shift	← Fim de sequência de 1s
1	0	Add 2's complement - Shift	← Início de sequência de 1s
1	1	Shift	

- Algoritmo baseia-se na decomposição de sequências de 1s:

$$\sum_{i=0}^{k-1} y_i = 2^k - 2^0 \qquad \text{Exemplo: } 1111 = 10000 - 0001$$

- Algoritmo para multiplicador de n-bits: i = 0 … n; $i_{-1} = 0$

Algoritmo de Booth "2 bits at a time"

y_{i+1}	y_i	y_{i-1}	Operação	
0	0	0	Shift 2 bits	Sequência de 0s
0	0	1	Add X – shift 2 bits	Fim de sequência de 1s na posição (**i-1**)
0	1	0	Add X - shift 2 bits	1 isolado na posição **i**
0	1	1	Add 2*X - shift 2 bits	Fim de sequência de 1s na posição **i**
1	0	0	Sub 2*X - shift 2 bits	Início de sequência de 1s na posição (**i+1**)
1	0	1	Sub X - shift 2 bits	Fim de sequência de 1s na posição (**i-1**) e Início de sequência de 1s na posição (**i+1**)
1	1	0	Sub X - shift 2 bits	Início de sequência de 1s na posição **i**
1	1	1	Shift 2 bits	Sequência de 1s

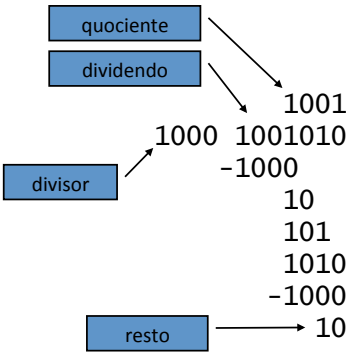
Metade do número de Produtos Parciais gerados – **n/2 ciclos**

Multiplicação no MIPS

- Dois registos de 32-bit para o produto
  - HI: most-significant 32 bits
  - LO: least-significant 32-bits
- Instruções
  - `mult rs, rt  /  multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd  /  mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits
  - `mul rd, rs, rt     # pseudo-instrução`
    - Least-significant 32 bits of product –> rd

Divisão

Dividendo = Quociente * Divisor + Resto

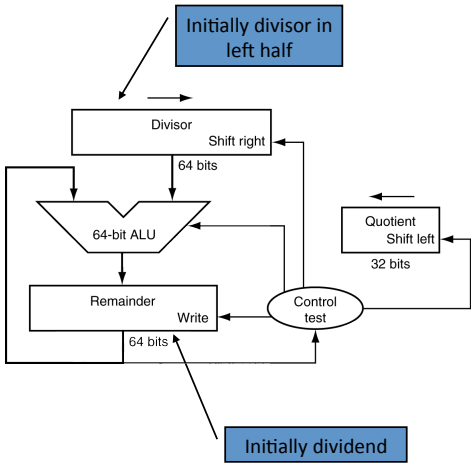


Operandos de n-bits: *n*-bit quociente e resto: *n*-bit

Dividendo = Quociente * Divisor + Resto

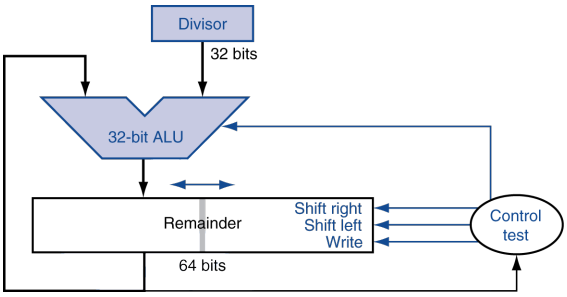
- Check for 0 divisor
- Divisor ≠ 0
  - ***If*** divisor ≤ bits do dividendo
    - novo bit do quociente = 1, subtrair
  - ***else***
    - novo bit do quociente = 0
  - Juntar ao resto parcial bit seguinte do dividendo
- ***Restoring division***
  - Subtrair sempre o divisor; se resto < 0 somar de novo o divisor
- Signed division
  - Dividir usando os valores absolutos
  - Ajustar o sinal do quociente e do resto

Divisor



Restoring Division

Divisor otimizado



- Um ciclo por cada subtração do resto parcial
- Semelhante ao multiplicador!
  - O mesmo hardware (com diferente algoritmo de controlo) pode ser usado para ambos

Divisão no MIPS

- Registos HI/LO usados para o resultado
  - HI: resto da divisão
  - LO: quociente
- Instruções
  - `div rs, rt  /  divu rs, rt`
  - **No divide-by-0 checking**
    - Cabe ao software fazer o teste, se necessário
  - `mfhi`, `mflo` para aceder ao resultado