para i variando de 0 até $n/4 - 1$

$A = p_{4i+n+2..4i+n-1}$
$B = x_{4i+3..4i}$
$(C, p_{4i+n+2..4i+n-1}) = A + \overline{B}\ and\ K + C$

$p_{2n-1} = C$
$C = K = x_{n-1}$

para i variando de 0 até $n/4 - 1$

$A = p_{4i+n+3..4i+n}$
$B = y_{4i+3..4i}$
$(C, p_{4i+n+3..4i+n}) = A + \overline{B}\ and\ K + C$

Este algoritmo, da forma como está exposto acima, apresenta um problema para a sua implementação. Os dados são lidos e escritos na memória de 4 em 4 *bits*. Para isto, basta que a memória tenha largura de 4 *bits*. Porém, tanto a leitura quanto a escrita dos resultados parciais nem sempre são executadas sobre o mesmo conjunto de 4 *bits*. Por exemplo, na atribuição inicial, p é dividido em palavras de 4 *bits* a partir do *bit* 0. Já na primeira soma, p é dividido em palavras de 4 *bits* a partir do *bit* 1. Isto ocorre porque a multiplicação é composta por um conjunto de somas com deslocamento de apenas um *bit*. Sendo assim, a primeira soma (substituída pela atribuição inicial) começa a partir do *bit* 0, a segunda soma começa a partir do *bit* 1 e assim por diante. Implementar uma memória em que se possa ter acesso a qualquer subconjunto de 4 *bits* não é uma tarefa muito simples, muito menos de baixo custo. Logo, faz-se necessário encontrar outra solução. A solução mais óbvia é a utilização de um *barrel shifter* para deslocar um dos operandos. Um *barrel shifter* é um dispositivo cuja saída é dada pela entrada deslocada para esquerda ou para a direita, quantas posições forem necessárias. O custo desta solução em termos de desempenho é o aumento do número de somas. Quando uma soma necessita de algum deslocamento, ela é subdividida em duas subsomas. Por exemplo, considere a seguinte soma de 4 *bits*: $p_{4..1} = p_{4..1} + x_{3..0}$, onde tanto p quanto x estão armazenados na memória a partir do *bit* 0, como é mostrado na Figura III.11. Pode-se dividir a soma em duas etapas como é mostrado a seguir:

$(C, p_{3..0}) = p_{3..0} + (x_{3..0}\ shift\ left\ 1)$
$p_{7..4} = p_{7..4} + (x_{3..0}\ shift\ rigth\ 3) + C$

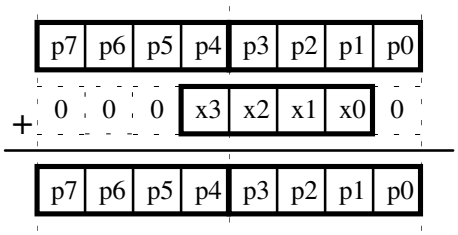

Figura III.10 - Soma de uma palavra de 4 *bits* deslocada de um *bit*

Observe que, embora as duas subsomas resolvam a questão, elas não são absolutamente equivalentes a soma inicial. Na soma inicial, os *bits* 5, 6 e 7 de p ficam inalterados após a soma.

Já nas duas subsomas, o *carry* é propagado até o *bit* 7 de *p*. Portanto, ao se utilizar esta substituição, deve-se tomar cuidado com a propagação do *carry* e o seu posicionamento.

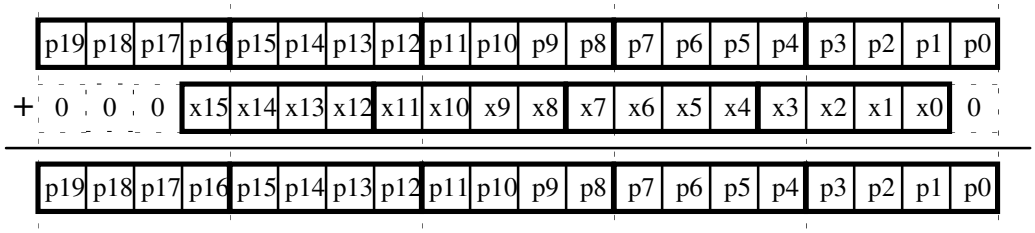

Figura III.11 - Soma deslocada de um *bit* para memória com largura de 4 *bits*

A soma $p_{17..1} = p_{16..1} + x_{15..0}$ é mostrada na Figura III.11. Um conjunto de subsomas que resolve esta soma é:

$$(c_4, p_{3..0}) = p_{3..0} + (x_{3..0}\ \textit{shift left}\ 1)$$
$$(c_8, p_{7..4}) = p_{7..4} + (x_{7..4}\ \textit{shift left}\ 1) + c_4$$
$$(c_{12}, p_{11..8}) = p_{11..8} + (x_{11..8}\ \textit{shift left}\ 1) + c_8$$
$$(c_{16}, p_{15..12}) = p_{15..12} + (x_{15..12}\ \textit{shift left}\ 1) + c_{12}$$
$$(c_8, p_{7..4}) = p_{7..4} + (x_{3..0}\ \textit{shift rigth}\ 3)$$
$$(c_{12}, p_{11..8}) = p_{11..8} + (x_{7..4}\ \textit{shift rigth}\ 3) + c_8$$
$$(c_{16}, p_{15..12}) = p_{15..12} + (x_{11..8}\ \textit{shift rigth}\ 3) + c_{12}$$
$$(c_{20}, p_{19..16}) = p_{19..16} + (x_{15..12}\ \textit{shift rigth}\ 3) + c_{16}$$

Numa implementação em que só existem dois registradores para o armazenamento dos operandos, esta solução tem a desvantagem de exigir a leitura e escrita de cada palavra de *p* e de *x* duas vezes. Desta forma o custo da soma é dobrado em relação a uma soma sem deslocamento. É possível amenizar este problema rearrumando as subsomas, como é mostrado abaixo:

$$(c_4, p_{3..0}) = p_{3..0} + (x_{3..0}\ \textit{shift left}\ 1)$$
$$(c_8, p_{7..4}) = p_{7..4} + (x_{3..0}\ \textit{shift rigth}\ 3) + c_4$$
$$(c_8, p_{7..4}) = p_{7..4} + (x_{7..4}\ \textit{shift left}\ 1) + (c_8, 0000)$$
$$(c_{12}, p_{11..8}) = p_{11..8} + (x_{7..4}\ \textit{shift rigth}\ 3) + c_8$$
$$(c_{12}, p_{11..8}) = p_{11..8} + (x_{11..8}\ \textit{shift left}\ 1) + (c_{12}, 0000)$$
$$(c_{16}, p_{15..12}) = p_{15..12} + (x_{11..8}\ \textit{shift rigth}\ 3) + c_{12}$$
$$(c_{16}, p_{15..12}) = p_{15..12} + (x_{15..12}\ \textit{shift left}\ 1) + (c_{16}, 0000)$$
$$(c_{20}, p_{19..16}) = p_{19..16} + (x_{15..12}\ \textit{shift rigth}\ 3) + c_{16}$$

Observe que nem sempre o *carry* é propagado para o *bit* menos significativo da próxima subsoma. Isto ocorre porque nem sempre a próxima subsoma é executada sobre um novo subconjunto de *p*. Para implementar esta propagação diferente do *carry*, é necessário fazer uma pequena alteração no somador. O somador proposto tem duas entradas de 4 *bits* (para os dois operandos) e duas entradas de *carry* (uma no *bit* 0 e outra no *bit* 4). Para facilitar a implementação, apenas uma delas pode ser utilizada de cada vez, ou seja, um *bit* de controle

indica em que posição entra o *carry*. Para apresentar os algoritmos será utilizada a notação de concatenação $(C,0000)$, que indica que C está direcionado para a entrada de *carry* no *bit* 4.

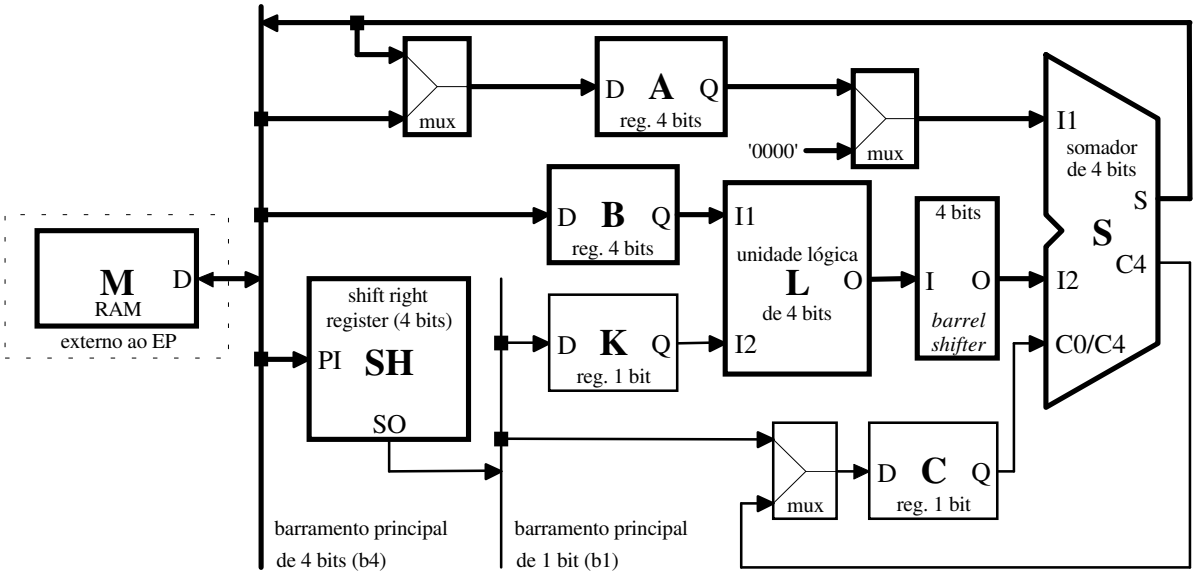

Figura III.12 - EP básico de 4 *bits* para a multiplicação

Se forem utilizadas duas variáveis auxiliares A e B, que equivalem aos registradores de armazenamento dos operandos no processador, pode-se chegar a um algoritmo geral de soma com deslocamento. Seja a soma $p_{4j+h+n-4j+h} = p_{4j+h+n-4j+h} + x_{4i+n-4i}$., onde $0 \le h < 4$. O algoritmo que resolve esta soma é:

$A = P_j, C = 0$
$B = X_i$
para k variando de 1 até $n/4-1$
$(C, P_{j+k-1}) = A + (B \textit{ shift left } h) + (C, 0000)$
$A = P_{j+k}$
$(C, A) = A + (B \textit{ shift right } (4 - h)) + C, B = X_{i+k}$
$(C, P_{j+n/4-1}) = A + (B \textit{ shift left } h) + (C, 0000)$
$A = P_{j+n/4}$
$P_{j+n/4} = A + (B \textit{ shift right } (4 - h))$

A soma apresentada acima pode ser processada em $3n/4+2$ ciclos, apenas um ciclo a mais do que uma soma sem deslocamento.

A Figura III.12 apresenta um EP básico de 4 *bits*. O bloco de 4 *bits* do EP possui dois registradores de 4 *bits* (A e B) para o armazenamento de operandos, uma unidade lógica de 4 *bits* (L), um somador de 4 *bits* com duas entradas de *carry* (como foi explicado anteriormente) e um *barrel shifter* capaz de deslocar um operando de 4 *bits* até quatro posições para a esquerda ou para a direita. O bloco de 1 *bit* do EP possui um registrador de 1 *bit* para o armazenamento do *carry* (C) e um registrador de 1 *bit* para as operações condicionais (K).

Este último entra, replicado 4 vezes, como segundo operando da unidade lógica (L). Para permitir a comunicação entre o bloco de 4 *bits* e o bloco de 1 *bit* do EP, é utilizado um registrador de deslocamento de 4 *bits* (SH), cuja entrada paralela está ligada ao barramento principal de 4 *bits* e a saída serial está ligada ao barramento principal de 1 *bit*.

O algoritmo que executa a multiplicação para este EP básico de 4 *bits* é apresentado no Apêndice A. Ele processa esta operação em $12(n/4)^2 + 10n/4 + 1$ ciclos. São executadas $n+1$ operações: uma atribuição, $n-2$ somas e duas subtrações (sendo que uma com deslocamento e outra sem deslocamento) envolvendo n *bits*. A atribuição consome $2n/4+1$ ciclos. Das somas, $n/4-1$ são sem deslocamento, enquanto que $3n/4-1$ são com deslocamento. Logo, são executadas $n/4$ somas ou subtrações sem deslocamento e $3n/4$ somas ou subtrações com deslocamento. Cada soma ou subtração consome $3n/4+2$ ciclos quando não há deslocamento e $3n/4+2$ ciclos quando há deslocamento. Os $n/4$ ciclos restantes são gastos na leitura do operando y. A maior parte do tempo de execução é consumido pelas somas que, além do problema do deslocamento de *bits*, sofrem com a falta de memória interna.

III.5.6 - EP de 4 *Bits* com Banco de Registradores para a Multiplicação

O desempenho do EP básico de 4 *bits* para multiplicação ficou aquém do desejável, considerando-se o custo associado a um processador de 4 *bits*. O superioridade desta arquitetura em relação aos EPs de 1 *bit* com dispositivos de armazenamento de dados internos é muito pequena.

A solução encontrada para resolver os problemas de falta de memória interna e dificuldade de operar dados deslocados é a substituição do registrador A por um banco de registradores A, contendo 8 registradores de deslocamento de 4 *bits* conectados entre si, formando um grande registrador de deslocamento de 32 *bits*, como mostra a Figura III.13. Somente as operações de deslocamento são realizadas em todos os registradores. Tanto as operações de leitura quanto as operações de escrita são realizadas em um único registrador de deslocamento de 4 *bits* de cada vez. Para selecionar qual dos 8 registradores será utilizado na leitura ou escrita, são empregados 3 *bits* de controle. Para fazer referência a um registrador específico do banco, seja para escrita ou para leitura, será utilizada a notação $A[i]$, que representa o *i*-ésimo registrador do banco A, como indicado na Figura III.13.

A Figura III.14 apresenta um EP de 4 *bits* com banco de registradores. Ele tem uma estrutura bem similar a do processador de 4 *bits* apresentado na seção anterior. O bloco de 4 *bits* do EP possui um banco de registradores (A) para o armazenamento de resultados parciais e um registrador de 4 *bits* (B) para o armazenamento de operandos, uma unidade lógica de 4 *bits* (L) e um somador de 4 *bits*. O bloco de 1 *bit* do EP possui um registrador de 1 *bit* para o armazenamento do *carry* (C) e um registrador de 1 *bit* para as operações condicionais (K).

Este último entra, replicado 4 vezes, como segundo operando da unidade lógica (L). Para permitir a comunicação entre o bloco de 4 *bits* e o bloco de 1 *bit* do EP, é utilizado um registrador de deslocamento de 4 *bits* (SH).

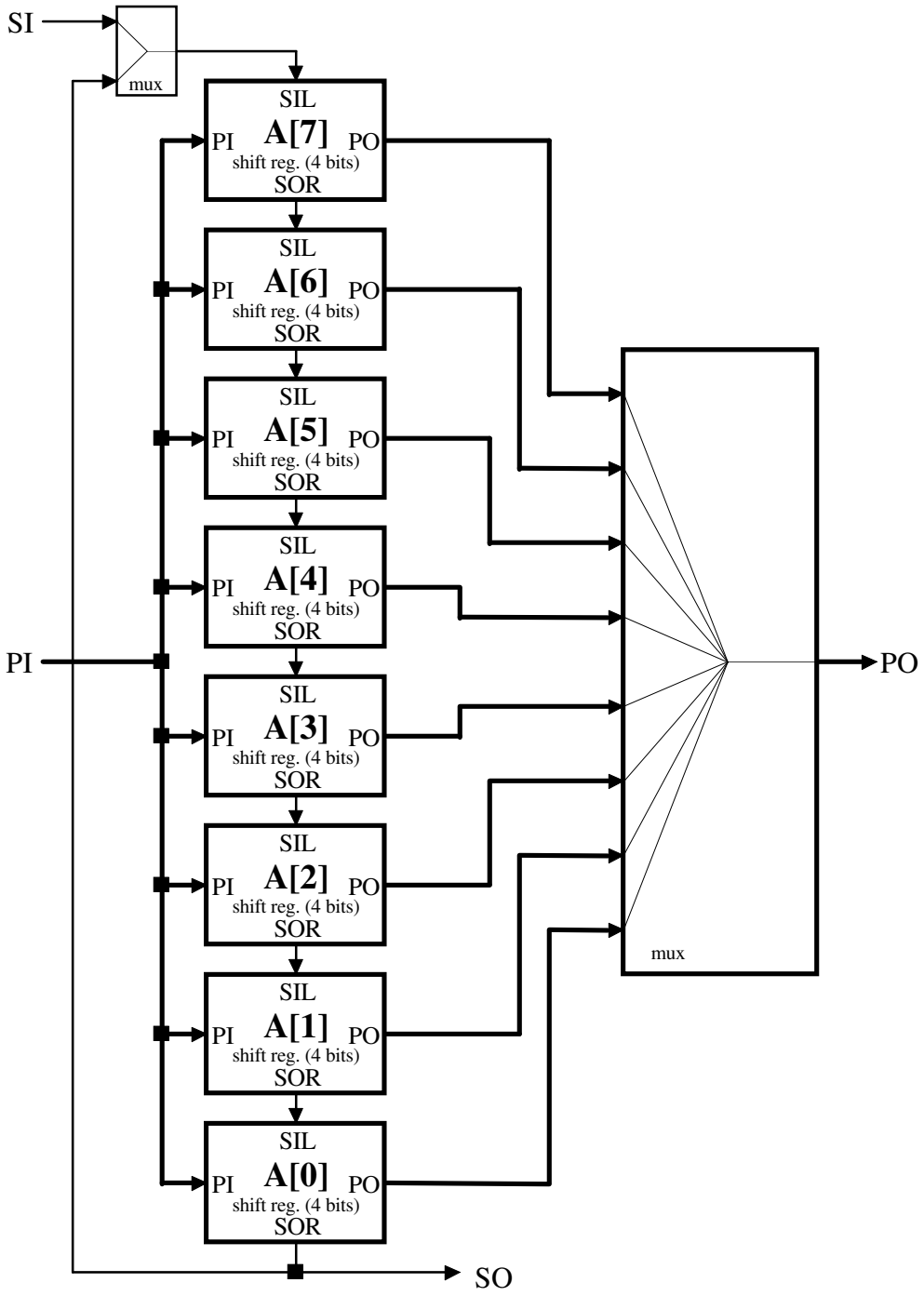

Figura III.13 - Banco de registradores A para a multiplicação

Para este Elemento Processador foram desenvolvidos dois algoritmos de multiplicação (Apêndice A): um para fatores de até 28 *bits* e outro para fatores de 32 *bits*. Esta separação é uma decorrência da escolha da capacidade de armazenamento do banco de registradores, que é de apenas 32 *bits*. Para multiplicações com fatores de 32 *bits* são necessários até 36 *bits* para

armazenamento de resultados parciais. Para contornar este problema, nestas multiplicações, o registrador de deslocamento SH é utilizado como dispositivo de armazenamento auxiliar dos resultados parciais.

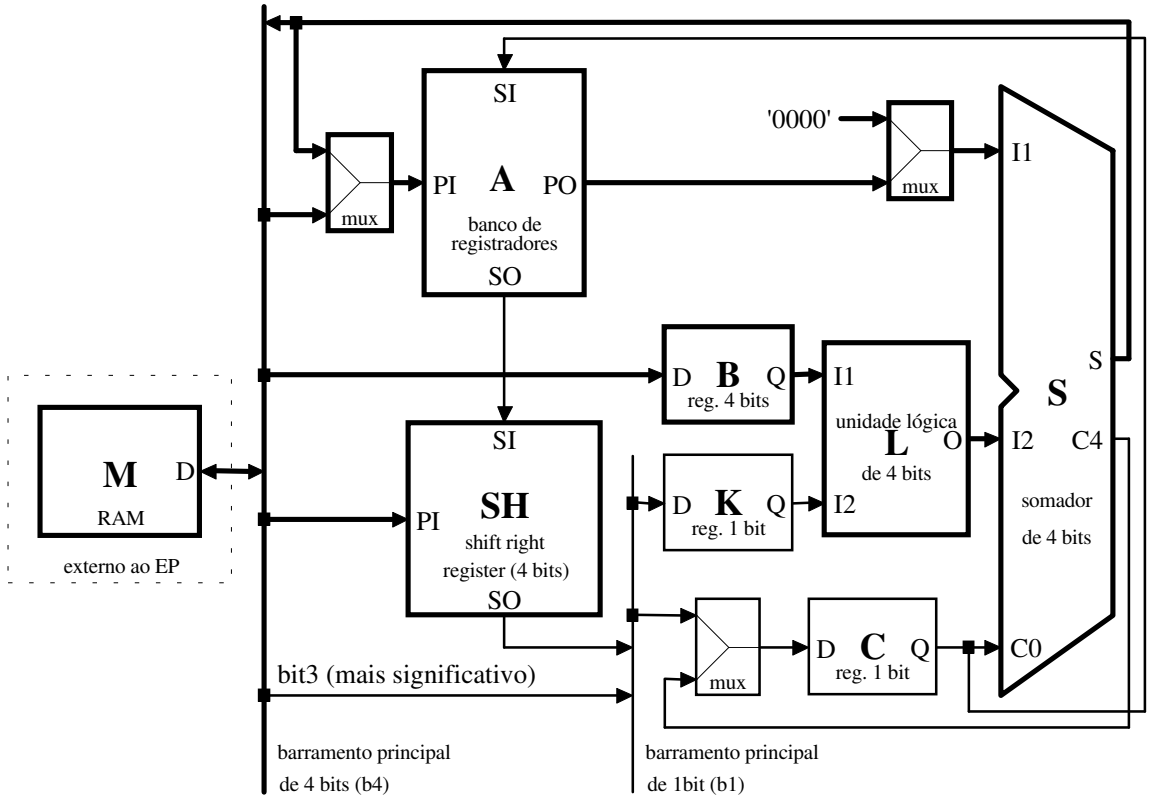

Figura III.14 - EP de 4 *bits* com banco de registradores para a multiplicação

O algoritmo para fatores de até 28 *bits* processa a multiplicação em $4(n/4)^2 + 7n/4 + 2$ ciclos. São executadas $n+2$ operações: uma atribuição, $n-2$ somas e duas subtrações de *n bits*, além de uma atribuição de 2*n bits*). A atribuição de *n bits* consome $n/4+1$ ciclos e a atribuição de 2*n bits* consome $2n/4$ ciclos. Cada soma ou subtração consome $n/4+1$ ciclos. O ciclo restante é gasto na leitura da primeira palavra do operando *y*.

O algoritmo para fatores de 32 *bits* processa a multiplicação em $4(n/4)^2 + 9n/4$ ciclos. A diferença de $2n/4-2$ ciclos em relação ao algoritmo anterior está na leitura do operando y, que não pode mais ser executada previamente e em paralelo com uma soma.

O desempenho deste EP de 4 *bits* com banco de registradores na execução da multiplicação é claramente superior ao desempenho de todas as outras propostas de arquiteturas apresentadas. Além disto, relativamente ao seu desempenho, o seu custo não é excessivamente elevado, como será visto mais adiante.

III.6 - Divisão de Inteiros

III.6.1 - Divisão Com Restauração e Sem Restauração

Assim como na multiplicação, a análise da divisão será separada em duas partes: divisão de inteiros positivos e divisão de inteiros com sinal.

Considere a divisão de *x* por *y*, tendo como resultados o quociente *q* e o resto *r*, onde *x*, *y*, *q* e *r* são números inteiros positivos representados por *n bits*. Pode-se afirmar que:

$$x = q \times y + r, \text{ onde } 0 \le r < y$$

Um algoritmo intuitivo para executar a divisão de *x* por *y* é:

x' = expansão de x para $2n-1$ bits
para *i* variando de 1 até *n*
    se $x'_{2n-1-i..n-i} \ge y$, então
        $q_{n-i} = 1$
        $x'_{2n-1-i..n-i} = x'_{2n-1-i..n-i} - y$
    senão
        $q_{n-i} = 0$
$r = x'_{n-1..0}$

Reescrevendo o algoritmo de uma outra forma:

x' = expansão de x para $2n-1$ bits
para *i* variando de 1 até *n*
    $x' = x' - 2^{n-i} y$
    se $x' \ge 0$, então
        $q_{n-i} = 1$
    senão
        $q_{n-i} = 0$
        $x' = x' + 2^{n-i} y$
$r = x'_{n-1..0}$

Este algoritmo é denominado divisão com restauração (*restoring division*). Para fazer a comparação, x' é subtraído de $2^{n-i} y$. Esta subtração adianta a operação que tem que ser executada caso a comparação tenha resultado positivo. Por outro lado, se a comparação tiver resultado negativo, a subtração não deve ser executada. Neste caso, o valor de x' tem que ser restaurado através de uma soma. Daí vem o nome divisão com restauração. Para executar uma divisão através deste algoritmo são necessárias: uma atribuição de $2n-1$ *bits*, *n* comparações, *n* somas e *n* subtrações de *n bits*, *n* atribuições de 1 *bit* e uma atribuição de *n bits*.

Através do uso de uma variável auxiliar é possível fazer as restaurações com operações de atribuição, modificando ligeiramente o algoritmo:

$x' =$ expansão de x para $2n-1$ bits
para i variando de 1 até n
    $z = x' - 2^{n-i} y$
    se $z \geq 0$, então
        $q_{n-i} = 1$
        $x' = z$
    senão
        $q_{n-i} = 0$
$r = x'_{n-1\cdots 0}$

Para executar uma divisão através deste algoritmo são necessárias: uma atribuição de $2n-1$ *bits*, n comparações, n subtrações de n *bits*, n atribuições de 1 *bit* e $n+1$ atribuições de n *bits*.

Uma forma mais eficiente de eliminar as operações de soma é fazer uma compensação na próxima subtração. Considere que, para $i = k$, $x' - 2^{n-i} y < 0$, ou seja, $x' - 2^{n-k} y < 0$. A restauração é feita através da operação $x' = x' + 2^{n-k} y$: Na iteração seguinte ($i = k+1$) é executada a operação $x' = x' - 2^{n-i} y$, ou seja, $x' = x' - 2^{n-k-1} y$. Juntando as duas operações tem-se:

$$x' = x' + 2^{n-k} y - 2^{n-k-1} y = x' + 2^{n-k-1} y(2-1) = x' + 2^{n-k-1} y$$

Logo, a restauração pode ser feita simplesmente invertendo o sinal da próxima subtração, gerando uma soma. Utilizando-se este artifício, pode-se modificar o algoritmo para:

$x' =$ expansão de x para $2n-1$ bits
$sinal = +1$
para i variando de 1 até n
    $x' = x' - sinal.2^{n-i} y$
    se $x' \geq 0$, então
        $q_{n-i} = 1$
        $sinal = +1$
    senão
        $q_{n-i} = 0$
        $sinal = -1$
se $sinal == +1$, então
    $r = x'_{n-1\cdots 0}$
senão
    $r = x'_{n-1\cdots 0} + y$

Este algoritmo é denominado divisão sem restauração (*non-restoring division*). Ele é claramente superior ao algoritmo de divisão com restauração. Para executar uma divisão através deste algoritmo são necessárias: uma atribuição de $2n-1$ *bits*, n comparações, $n+1$ somas ou subtrações de n *bits* e $2n+1$ atribuições de 1 *bit*.

III.6.2 - Divisão com Sinal

A primeira diferença entre as divisões de inteiros positivos e com sinal está no tamanho do resultado. Considere a divisão de *x* por *y*, tendo como resultados o quociente *q* e o resto *r*, onde *x*, *y*, *q* e *r* são números inteiros representados em complemento a dois. Considere ainda que *x* e *y* são representados por *n bits*. O resto *r* é menor do que *y*, logo ele também pode ser representado por *n bits*. A diferença se encontra no quociente. O maior valor absoluto para o quociente ocorre para a divisão entre o maior valor absoluto de *x* e o menor valor absoluto (diferente de zero) de *y*. Ou seja, o maior valor absoluto do quociente é $2^n (= 2^n / 1)$. Porém, este valor só pode ser representado por *n bits* se o seu sinal for negativo. Para a divisão $-2^n / -1$, o quociente é $+2^n$, o que significa que ele precisa ser representado por pelo menos $n+1$ *bits*.

A maneira mais óbvia de resolver o problema da divisão de inteiros com sinal é calcular os módulos do dividendo e do divisor, executar uma divisão sem sinal usando os módulos e, depois, calcular os sinais do quociente e do resto. Aproveitando o algoritmo de divisão sem restauração, chega-se a:

$x' =$ expansão de $|x|$ para $2n-1$ bits
$y' = |y|$
$sinal = +1$
para *i* variando de 1 até *n*
    $x' = x' - sinal.2^{n-i}\, y'$
    se $x' \geq 0$, então
        $q_{n-i} = 1$
        $sinal = +1$
    senão
        $q_{n-i} = 0$
        $sinal = -1$
se $sinal == +1$, então
    $r = x'_{n-1\cdots 0}$
senão
    $r = x'_{n-1\cdots 0} + y'$
se $x < 0$, então
    $r = -r$
se *x* e *y* têm sinais opostos, então
    $q = -q$

Para executar uma divisão através deste algoritmo são necessários: um cálculo de módulo de $2n-1$ *bits*, e outro de *n bits*, $n+3$ comparações, $n+1$ somas ou subtrações de *n bits*, $2n+1$ atribuições de 1 *bit* e duas inversões de sinal de *n bits*.

Existem outros algoritmos para divisão com sinal, também derivados do princípio da divisão sem restauração, que dispensam o cálculo dos módulos do dividendo e/ou do divisor. Porém, eles necessitam de operações condicionais mais complexas. Isto significa que, para

aproveitar todas as suas potencialidades a arquitetura deve dispor de mais dispositivos de mascaramento. Além disso, ao final do laço principal do algoritmo, a mesma quantidade de correções é necessária. Logo, não há vantagem no uso destes algoritmos.

A implementação do algoritmo apresentado depende da arquitetura utilizada. Cada caso será analisado detalhadamente mais adiante. Porém, algumas considerações de caráter geral podem ser apresentadas desde já.

A primeira se refere ao mapeamento das variáveis auxiliares x' e y' na memória. Observando o algoritmo, pode-se perceber que a primeira vez em que é atribuído um valor ao resto *r* corresponde a última vez em que é utilizada a variável y'. Logo, a variável auxiliar y' pode ser mapeada na mesma região de memória reservada para o resto *r*.

Quanto a variável x', o seu mapeamento é dependente da arquitetura. Se o processador não possui memória interna, uma região da memória externa deve ser alocada para o mapeamento de variáveis auxiliares de cálculo. Este espaço deve ser reservado para o uso exclusivo do processador e o seu endereço será denominado *z*. Deve-se tomar cuidado para que não seja permitido ao usuário alocar variáveis nesta região. Por outro lado, se o processador possui memória interna, a variável x' pode ser alocada nela. Nem sempre a memória interna é suficiente para alocar toda a variável x'. Quando isto ocorre, a solução é alocar os *bits* mais significativos de x' (os primeiros a serem utilizados no algoritmo) na memória interna e utilizar a região de memória reservada para o quociente *q* para armazenar os *bits* menos significativos. Esta solução só é possível porque, para cada *bit* do quociente *q* calculado, pode-se dispensar um *bit* de x' (o mais significativo) e, conseqüentemente, transportar um *bit* de x' da região reservada ao quociente *q* para a memória interna.

Outra consideração importante se refere a variável *sinal*. A ela só podem ser atribuídos dois valores: +1 e -1. Logo, um único *bit* é capaz de representá-la. Resta decidir quem vai implementá-la e quais serão as representações para estes dois valores. Para implementá-la pode-se utilizar o registrador de máscara K, presente em todas as arquiteturas apresentadas para a multiplicação. Quanto a representação, observe que, após cada atribuição de um *bit* ao quociente *q*, uma atribuição de um *bit* é feita à variável *sinal*. Sendo assim, a fim de agilizar o algoritmo, deve-se escolher uma representação tal que o mesmo valor seja atribuído ao quociente *q* e à variável *sinal*. Ou seja, para o algoritmo apresentado:

$K = 0$ corresponde a $sinal = -1$

$K = 1$ corresponde a $sinal = +1$

Por último, uma análise mais cuidadosa permite concluir que o cálculo da extensão do módulo de *x* para $2n-1$ *bits* não é necessário. Como *x* é representado por *n bits*, o seu módulo também pode ser representado por *n bits*. Portanto, a extensão do módulo, ou seja, os $n-1$ *bits*

restantes, serão iguais a zero. Se o resultado da extensão já é conhecido de antemão, o seu cálculo não é necessário. É mais eficiente calcular o módulo de *x* com apenas *n bits*. Na primeira iteração do laço principal do algoritmo, os $n-1$ *bits* restantes serão necessários na subtração. Neste momento, basta utilizar o valor zero como operando da subtração. Para isto a primeira iteração será separada do laço principal.

A partir das análises apresentadas acima, pode-se deduzir três algoritmos básicos para a execução da divisão entre inteiros com sinal: um para processadores sem memória interna, uma para processadores com memória interna de pelo menos *n bits* e um para processadores com memória interna de pelo menos 2*n bits*.

Para processadores sem dispositivos de memória interna (variável *x′* mapeada na memória externa, no endereço *z*) o algoritmo de divisão é:

$z_{n-1..0} = |x|$
$r = |y|$
$K = C = 1$
$z_{2n-2..n-1} = (00\ldots0, z_{n-1}) + (r_{n-1..0}\ xor\ K) + C$
se $z_{2n-2..n-1} \geq 0$, então
    $q_{n-1} = K = C = 1$
senão
    $q_{n-1} = K = C = 0$
para *i* variando de 2 até *n*
    $z_{2n-i-1..n-i} = z_{2n-i-1..n-i} + (r_{n-1..0}\ xor\ K) + C$
    se $z_{2n-i-1..n-i} \geq 0$, então
        $q_{n-i} = K = C = 1$
    senão
        $q_{n-i} = K = C = 0$
$r_{n-1..0} = z_{n-1..0} + (r_{n-1..0}\ and\ \overline{K})$
$K = C = x_{n-1}$
$r_{n-1..0} = (r_{n-1..0}\ xor\ K) + C$
$K = C = K\ xor\ y_{n-1}$
$q_{n..0} = (q_{n-1..0}\ xor\ K) + C$

Considere agora os processadores com dispositivos de memória interna limitados, ou seja, com pelo menos *n bits* disponíveis, porém menos do que 2*n bits*. Nestes casos, apenas os *bits* mais significativos de *x′* (os primeiros a serem utilizados no algoritmo) são alocados na memória interna. Os *bits* menos significativos são armazenados na região da memória reservada para o quociente *q*. Considere *mi* a denominação da memória interna do Elemento Processador. O algoritmo genérico para a execução da divisão nestes processadores é:

$q_{n-1..0} = |x|$
$r = |y|$
$K = C = 1$
$mi_{n-1..0} = (00..00, q_{n-1}) + (r_{n-1..0}\ xor\ K) + C$

se $mi_{n-1..0} \geq 0$, então
    $q_{n-1} = K = C = 1$
senão
    $q_{n-1} = K = C = 0$
para *i* variando de 2 até *n*
    $mi_{n-1..0} = (mi_{n-2..0}, q_{n-i}) + (r_{n-1..0}\ xor\ K) + C$
    se $mi_{n-1..0} \geq 0$, então
        $q_{n-i} = K = C = 1$
    senão
        $q_{n-i} = K = C = 0$
$mi_{n-1..0} = mi_{n-1..0} + (r_{n-1..0}\ and\ \overline{K})$
$K = C = x_{n-1}$
$r_{n-1..0} = (mi_{n-1..0}\ xor\ K) + C$
$K = C = K\ xor\ y_{n-1}$
$q_{n..0} = (q_{n-1..0}\ xor\ K) + C$

Por último, considere os processadores com dispositivos de memória interna menos limitados, ou seja, com pelo menos $2n$ *bits* disponíveis. Nestes casos, a variável auxiliar x' é inteiramente alocada na memória interna. Considere *mi* a denominação da memória interna do EP. O algoritmo genérico para a execução da divisão nestes processadores é:

$mi_{n-1..0} = |x|$
$r = |y|$
$K = C = 1$
$mi_{2n-2..n-1} = (00..00, mi_{n-1}) + (r_{n-1..0}\ xor\ K) + C$
se $mi_{2n-2..n-1} \geq 0$, então
    $q_{n-1} = K = C = 1$
senão
    $q_{n-1} = K = C = 0$
para *i* variando de 2 até *n*
    $mi_{2n-i-1..n-i} = mi_{2n-i-1..n-i} + (r_{n-1..0}\ xor\ K) + C$
    se $mi_{2n-i-1..n-i} \geq 0$, então
        $q_{n-i} = K = C = 1$
    senão
        $q_{n-i} = K = C = 0$
$mi_{n-1..0} = mi_{n-1..0} + (r_{n-1..0}\ and\ \overline{K})$
$K = C = x_{n-1}$
$r_{n-1..0} = (mi_{n-1..0}\ xor\ K) + C$
$K = C = (y_{n-1..0}\ xor\ K)$
$q_{n..0} = (q_{n-1..0}\ xor\ K) + C$

Para executar uma divisão através de qualquer um dos três algoritmos apresentados acima, são necessários: dois cálculos de módulo de *n bits*, $n+1$ somas ou subtrações de *n bits*, $n+3$ atribuições de 1 *bit* e duas inversões de sinal de *n bits*. Apesar de as tarefas necessárias aos três algoritmos serem as mesmas, é natural esperar que o desempenho das suas

implementações seja diferenciado, visto que o acesso às memórias internas costuma ser mais eficiente do que o acesso às memórias externas.

III.6.3 - EP Básico de 1 *Bit* para a Divisão

Para implementar a divisão pode-se adaptar a arquitetura apresentada para a solução da multiplicação (Figura III.7), apenas acrescentando uma ligação entre a saída da unidade lógica (L) e o barramento principal de 1 *bit*, como mostra a Figura III.15. Esta ligação não é essencial, mas representa o ganho de alguns ciclos na implementação do algoritmo e tem um custo extremamente baixo. Além disto, ela pode ser útil na execução de outras operações.

O algoritmo de divisão para o EP da Figura III.15 (apresentado no Apêndice A), foi desenvolvido a partir do algoritmo para processadores sem memória interna apresentado na seção III.6.2. O seu desempenho está longe do ideal. Ele processa a divisão em $3n^2 + 10n + 6$ ciclos. São gastos $2n$ ciclos para o cálculo do módulo de x e mais $2n$ ciclos para o cálculo do módulo de y. Para calcular o sinal do quociente q são necessários apenas dois ciclos. A primeira subtração também gasta $2n$ ciclos. O laço principal executa $n-1$ iterações. Cada iteração é composta por uma soma ou subtração de n *bits* e uma atribuição de 1 *bit*, perfazendo um total de $3n$ ciclos por iteração. Por fim, as correções finais (uma soma condicional de n *bits* e duas inversões de sinal condicionais de n *bits*) consomem $7n + 4$ ciclos.

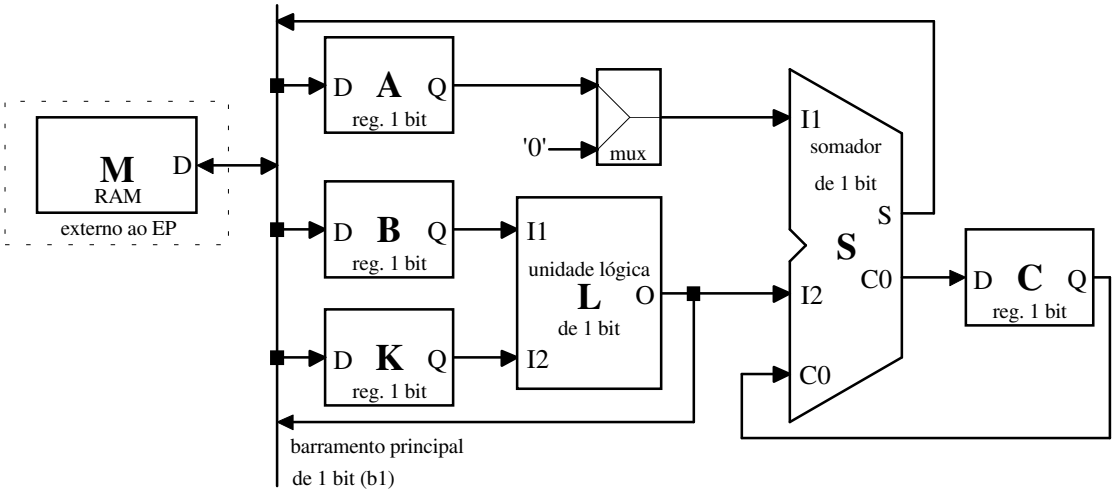

Figura III.15 - EP básico de 1 *bit* para a divisão

A análise do desempenho desta arquitetura para a divisão é bem parecida com a análise da multiplicação. A conclusão é a mesma: o gargalo do sistema é o acesso à memória. Sendo assim, a solução proposta também é a mesma, ou seja: criação de uma memória interna. A seguir, as duas implementações de dispositivos de memória interna propostas para a multiplicação serão analisadas para o caso da divisão.

III.6.4 - EP de 1 *Bit* com Memória Interna para a Divisão

A Figura III.16 apresenta o esquema de um elemento processador que é uma adaptação da arquitetura apresentada na Figura III.8 para a divisão. Este processador utiliza um registrador de deslocamento de tamanho programável (SH) para o armazenamento dos resultados parciais. No caso da divisão, ele é utilizado para o armazenamento de x'. Para uma melhor adaptação da arquitetura ao problema da divisão, foram acrescidas: uma ligação entre a saída do registrador B e o barramento principal de 1 *bit*, uma ligação entre o barramento principal de 1 *bit* e a entrada do registrador de *carry* (C) e uma ligação entre a saída do registrador A e a primeira entrada da unidade lógica (L). Nenhuma das modificações feitas é essencial, porém cada uma delas representa um ganho de alguns ciclos na implementação do algoritmo, sem grandes custos adicionais. A ligação entre o barramento principal e o registrador de *carry* (C), por exemplo, permite que o mesmo seja carregado ao mesmo tempo e com o mesmo valor que o registrador de máscara (K), sem grandes malabarismos, como no algoritmo apresentado para o EP básico de 1 *bit*. A diferença é que, para o caso anterior, o gargalo do sistema é o acesso à memória, conseqüentemente, uma modificação na arquitetura não representaria um ganho de tempo, o que não é o caso desta arquitetura. Já a ligação entre a saída do registrador A e a entrada da unidade lógica (L) é uma consequência da memória interna. Esta ligação tem por objetivo facilitar a inversão de sinal condicional do resto *r*, que está armazenado no registrador de deslocamento programável (SH).

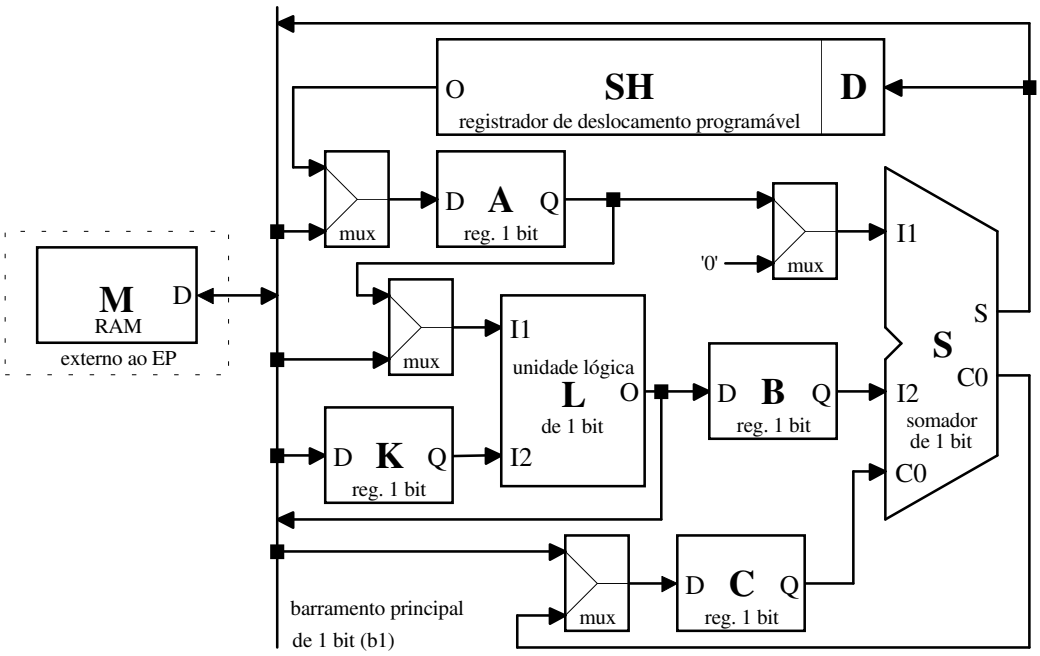

Figura III.16 - EP de 1 *bit* com registrador de deslocamento programável para a divisão

Quanto a implementação do algoritmo, esta arquitetura se encaixa no caso em que os *bits* mais significativos de x' são alocados na memória interna e os *bits* menos significativos

são armazenados na região da memória reservada para o quociente q. Ou seja, o algoritmo de divisão para o Elemento Processador de 1 *bit* com registrador de deslocamento programável (Apêndice A) foi desenvolvido a partir daquele apresentado na seção III.6.2 para estes casos.

O seu desempenho já é bem melhor do que o do EP básico de 1 *bit*. Ele processa a divisão em $n^2+10n+6$ ciclos. A etapa inicial é idêntica: são gastos $2n$ ciclos para o cálculo do módulo de x, $2n$ ciclos para o cálculo do módulo de y e dois ciclos para calcular o sinal do quociente q. A primeira subtração gasta $n+1$ ciclos. O laço principal executa $n-1$ iterações. Cada iteração é composta por uma soma ou subtração de n *bits* e uma atribuição de 1 *bit*, perfazendo um total de $n+2$ ciclos por iteração. Por fim, as correções finais (uma soma condicional de n *bits* e duas inversões de sinal condicionais de n *bits*) consomem $4n+5$ ciclos.

A Figura III.17 apresenta o esquema de um elemento processador que é uma adaptação da arquitetura apresentada na Figura III.9 para a divisão. Este processador utiliza um dispositivo de armazenamento, denominado A, similar a uma fila FIFO (*First In First Out*), para guardar os resultados parciais. No caso da divisão, ele é utilizado para o armazenamento de x'. Para uma melhor adaptação da arquitetura ao problema da divisão, foram acrescidas: uma ligação entre a saída do registrador B e o barramento principal de 1 *bit*, uma ligação entre o barramento principal de 1 *bit* e a entrada do registrador de *carry* (C) e uma ligação entre a saída da fila A e a primeira entrada da unidade lógica (L). As modificações são as mesmas feitas para o Elemento Processador de 1 *bit* com registrador de deslocamento programável, como era de se esperar, visto que as necessidades também são as mesmas.

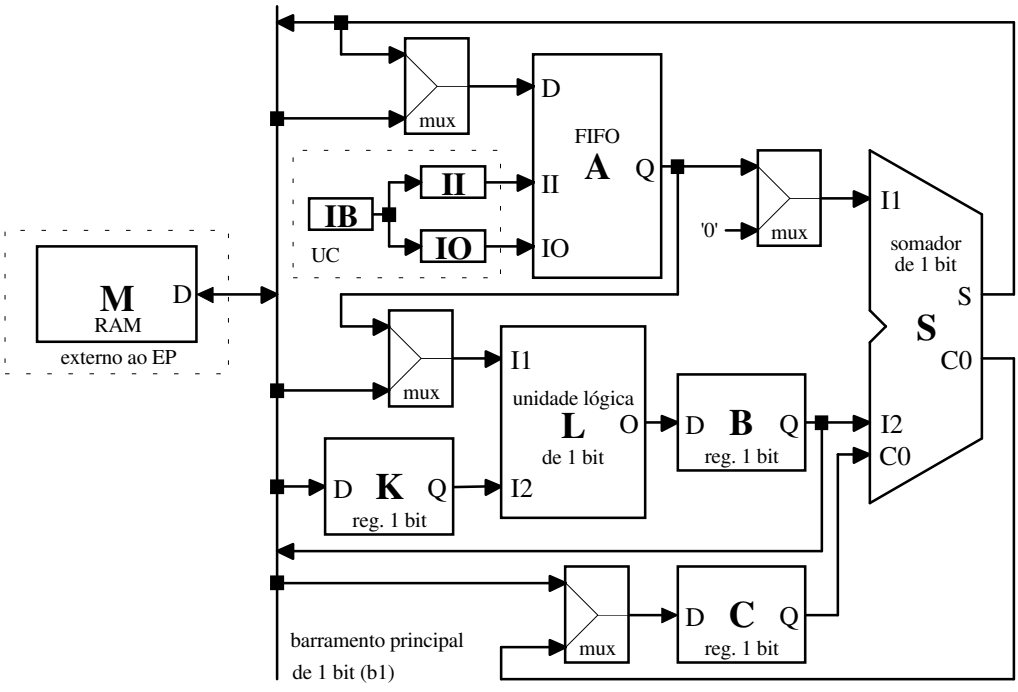

Figura III.17 - EP de 1 *bit* com fila FIFO para a divisão

Quanto ao algoritmo a ser utilizado, esta arquitetura se encaixa em dois casos, dependendo do tamanho dos operandos. Se os operandos tiverem mais do que 16 *bits*, só é possível armazenar os *bits* menos significativos de x' na região da memória reservada para o quociente *q*. Porém, se os operandos tiverem um máximo 16 *bits*, esta arquitetura permite o armazenamento completo de x' na memória interna. Sendo assim, foram desenvolvidos dois algoritmos de divisão para o Elemento Processador de 1 *bit* com fila FIFO, ambos apresentados no Apêndice A.

O desempenho do algoritmo para operandos com mais de 16 *bits* é praticamente igual ao do EP de 1 *bit* com registrador de deslocamento programável. Ele processa a divisão em $n^2+10n+7$ ciclos, ou seja, um ciclo a mais. Esta diferença de um ciclo se encontra exatamente no cálculo do módulo de *x*, que gasta $2n+1$ ao invés de $2n$ ciclos.

Já o desempenho do algoritmo para operandos de até 16 *bits* é um pouco melhor do que o do EP de 1 *bit* com registrador de deslocamento programável. Ele processa a divisão em $n^2+7n+10$ ciclos, ou seja, $3n-4$ ciclos a menos do que a outra arquitetura. Os ganhos se encontram em três etapas da divisão. Primeiramente no cálculo do módulo de *x*, que não precisa ser armazenado na memória externa, o que representa um ganho de $n-2$ ciclos. Depois, no laço principal, que não precisa buscar um *bit* do módulo de *x* na memória externa a cada iteração, o que economiza $n-1$ ciclos. Por último, na inversão condicional do sinal do quociente *q*, que não precisa ser lido da memória externa, economizando mais $n-1$ ciclos.

III.6.5 - EP Básico de 4 *Bits* para a Divisão

Assim como nas demais operações aritméticas, também na divisão, uma outra forma de melhorar o desempenho é trabalhar com dados de 4 *bits* ao invés de 1 *bit*. A Figura III.18 apresenta o esquema de um Elemento Processador que é uma adaptação da arquitetura apresentada na Figura III.10 para a multiplicação. Ela apresenta elementos e barramentos de 1 e 4 *bits*. Para uma melhor adaptação ao problema da divisão, fora feitas três modificações. A primeira foi a colocação de uma ligação entre a saída do *barrel shifter* e o barramento principal de 4 *bits*, com o intuito de economizar alguns ciclos. Esta modificação se assemelha às apresentadas para os EPs de 1 *bit* apresentados anteriormente. Também foi acrescentada uma ligação entre o *bit* mais significativo do barramento principal de 4 *bits* e a entrada do barramento principal de 1 *bit*. Esta modificação teve por objetivo agilizar o teste sobre o sinal do resultado das somas/subtrações do laço principal. Por último, o registrador de deslocamento para a direita (SH) foi substituído por um registrador bidirecional, permitindo uma comunicação entre o barramento de 4 *bits* e o barramento de 1 *bit*. No caso específico da divisão, este registrador de deslocamento facilita o armazenamento dos resultados parciais do quociente *q*.

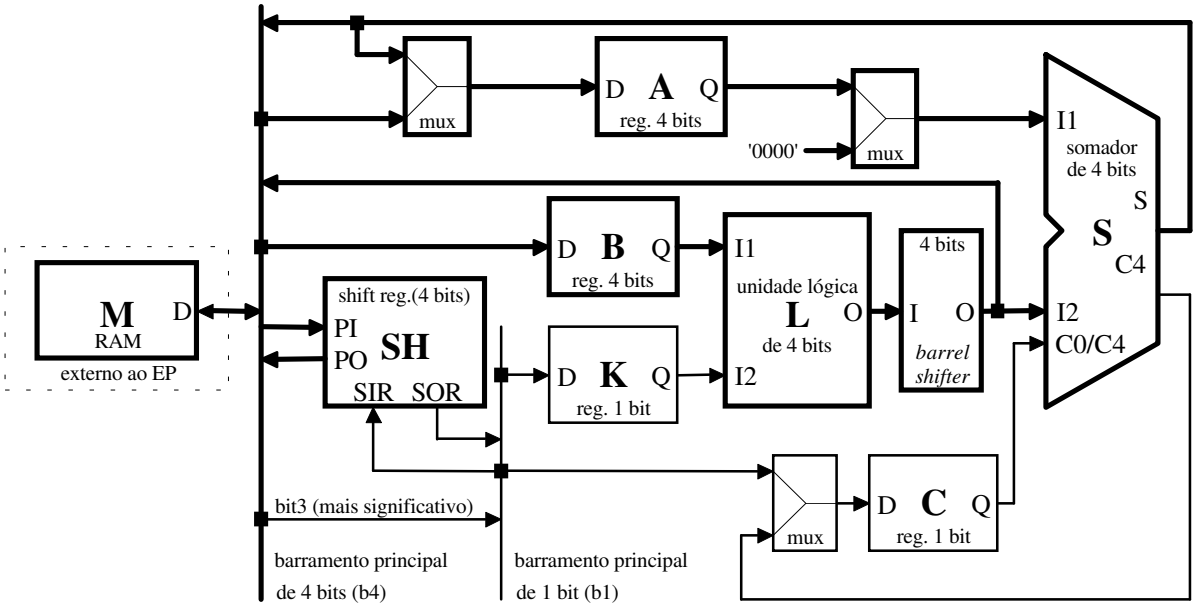

Figura III.18 - EP básico de 4 *bits* para a divisão

Quanto à implementação de um algoritmo de divisão para este processador, inicialmente vale ressaltar que o algoritmo desenvolvido para a implementação de somas entre variáveis armazenadas em memórias de 4 *bits* com deslocamento não múltiplo de 4, apresentado para a multiplicação (seção III.5.5), precisa sofrer uma pequena modificação para ser usado nas subtrações da divisão.

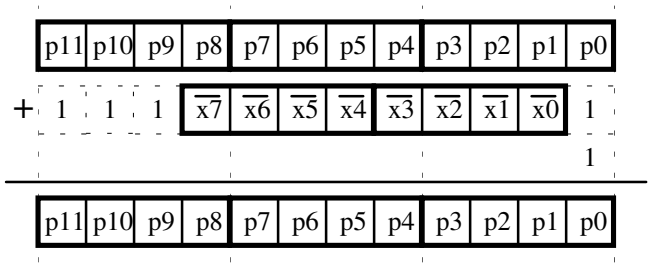

Figura III.19 - Subtração deslocada de um *bit* para memória com largura de 4 *bits*

A subtração $p_{11..1} = p_{10..1} - x_{7..0}$ é mostrada na Figura III.19. O conjunto de operações que resolve esta subtração é:

$(c_4, p_{3..0}) = p_{3..0} + (\overline{1111}\ shift\ right\ 3) + 1$
$(c_4, p_{3..0}) = p_{3..0} + (\overline{x_{3..0}}\ shift\ left\ 1) + (c_4, 0000)$
$(c_8, p_{7..4}) = p_{7..4} + (\overline{x_{3..0}}\ shift\ right\ 3) + c_4$
$(c_8, p_{7..4}) = p_{7..4} + (\overline{x_{7..4}}\ shift\ left\ 1) + (c_8, 0000)$
$(c_{12}, p_{11..8}) = p_{11..8} + (x_{7..4}\ shift\ right\ 3) + c_8$
$(c_{12}, p_{11..8}) = p_{11..8} + (1111\ shift\ left\ 1) + (c_{12}, 0000)$

Se forem utilizadas duas variáveis auxiliares *A* e *B*, que equivalem aos registradores de armazenamento dos operandos no processador, pode-se chegar a um algoritmo geral de

soma/subtração condicional com deslocamento. Seja a operação condicional abaixo, onde $0 \le h < 4$:

se $K == 1$, então
$p_{4j+h+n\cdot 4j+h} = p_{4j+h+n\cdot 4j+h} - x_{4i+n\cdot 4i}$.
else
$p_{4j+h+n\cdot 4j+h} = p_{4j+h+n\cdot 4j+h} + x_{4i+n\cdot 4i}$.

O algoritmo que resolve esta operação é:

$A = P_j, C = K$
$B = X_i, (C, A) = A + (K \; shift \; right \; (4-h)) + C$
para k variando de 1 até $n/4-1$
$(C, P_{j+k-1}) = A + ((B \; xor \; K) \; shift \; left \; h) + (C, 0000)$
$A = P_{j+k}$
$(C, A) = A + ((B \; xor \; K) \; shift \; right \; (4 - h)) + C, B = X_{i+k}$
$(C, P_{j+n/4-1}) = A + ((B \; xor \; K) \; shift \; left \; h) + (C, 0000)$
$A = P_{j+n/4}$
$(C, A) = A + ((B \; xor \; K) \; shift \; right \; (4-h)) + C$
$P_{j+n/4} = A + (K \; shift \; left \; h) + (C, 0000)$

Este algoritmo é processado em $3n/4+3$ ciclos, um a mais que a soma com deslocamento apresentada na seção III.5.5.

Como o processador apresentado não possui dispositivo de memória interna, o algoritmo de divisão desenvolvido (Apêndice A) baseou-se no algoritmo para processadores sem memória interna apresentado na seção III.6.2. O seu desempenho é ainda melhor do que o dos EPs de 1 *bit* com memória interna. Ele processa a divisão em $12(n/4)^2 + 25n/4 + 7$ ciclos. Inicialmente são gastos $3n/4+1$ ciclos para o cálculo do módulo de x estendido e $2n/4+1$ ciclos para o cálculo do módulo de y e um ciclo para atribuir um valor inicial aos registradores de máscara (K) e de *carry* (C). O laço principal executa n iterações. Destas, $3n/4$ iterações executam somas ou subtrações com deslocamento (ao custo de $3n/4+4$ ciclos cada iteração) e $n/4$ executam somas e subtrações sem deslocamento (ao custo de $3n/4+1$ ciclos cada iteração). Por fim, as correções finais (uma soma condicional de n *bits* e duas inversões de sinal condicionais de n *bits*) consomem $7n/4+4$ ciclos.

III.6.6 - EP de 4 *Bits* com Banco de Registradores para a Divisão

Assim como na multiplicação, o desempenho do EP básico de 4 *bits* ficou aquém do desejável, considerando-se o seu custo. Da mesma forma, a solução encontrada é a criação de um dispositivo de memória interna denominado banco de registradores, a exemplo do que foi sugerido na seção III.5.6. Algumas adaptações foram feitas no banco de registradores apresentado na Figura III.11. No caso da divisão, o deslocamento do conteúdo do banco de

registradores deve ser para a esquerda, ao contrário do que ocorre na multiplicação. Por isto, neste caso, o banco de registradores A contém 8 registradores de deslocamento **bidirecionais** de 4 *bits* conectados entre si, formando um registrador de deslocamento bidirecional de 32 *bits*, conforme mostra a Figura III.20. Esta modificação resultou na criação de mais uma entrada serial, utilizada nos deslocamentos para a esquerda, denominada SIR.

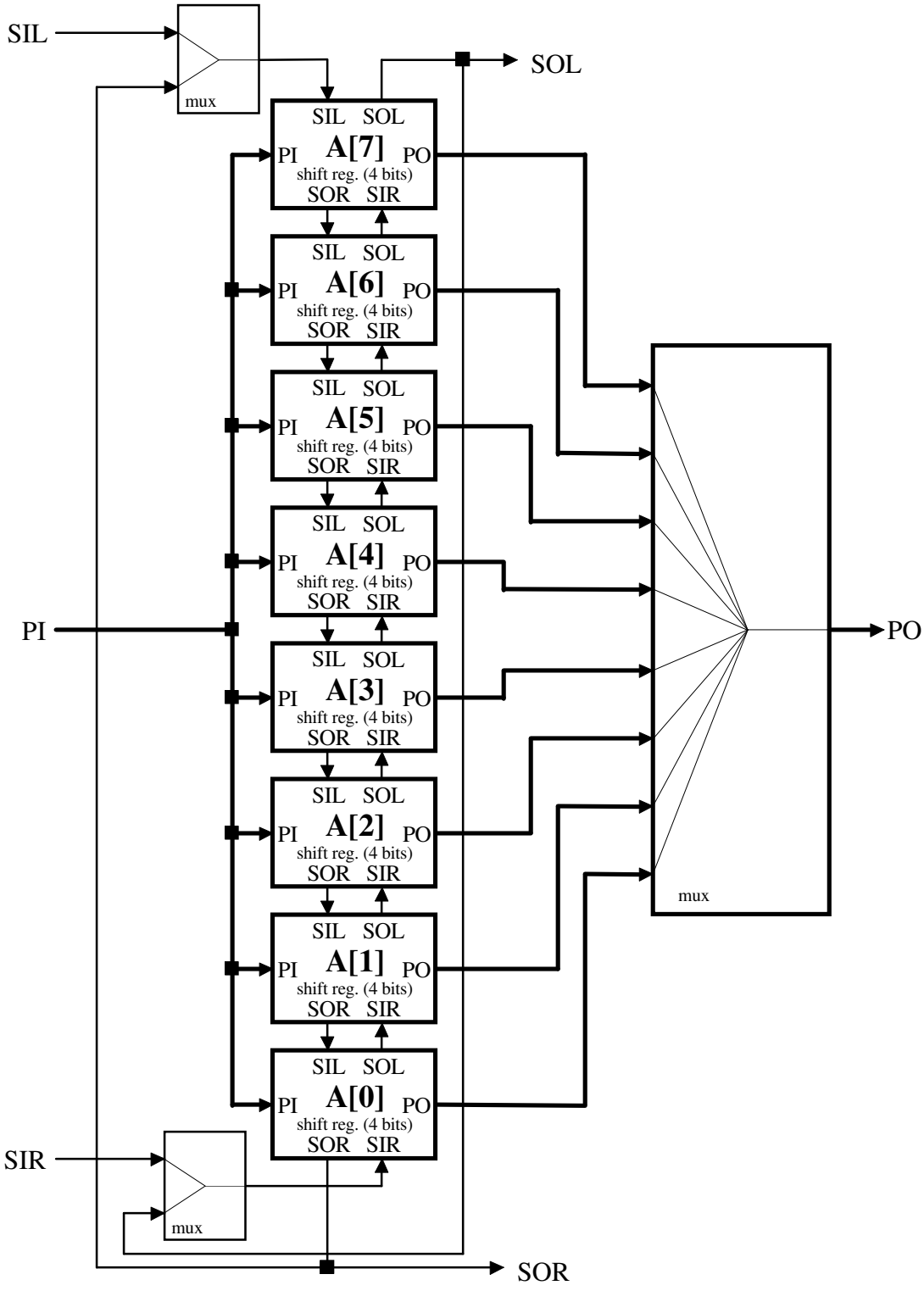

Figura III.20 - Banco de registradores A para a divisão

A Figura III.21 apresenta a versão do EP de 4 *bits* com banco de registradores para a divisão. Algumas modificações foram feitas em relação ao mesmo processador apresentado para a multiplicação (Figura III.12). Assim como o banco de registradores, também o registrador de deslocamento para a direita (SH) foi substituído por um registrador de deslocamento bidirecional. Sendo assim, o conjunto banco de registradores A, registrador de deslocamento SH pode ser visto como um grande registrador de deslocamento bidirecional de 36 *bits*. Além disto, foram acrescidas duas ligações: uma entre a saída da unidade lógica e o barramento principal de 4 *bits* e outra entre a saída paralela do banco de registradores (A) e a segunda entrada da unidade lógica. Estas modificações são similares e têm as mesmas motivações das apresentadas na seção III.6.4 para os processadores de 1 *bit* com memória interna. Por último, foi acrescentada uma entrada no barramento principal de 1 *bit*, correspondente ao valor do *bit* mais significativo do barramento principal de 4 *bits* invertido. Esta ligação tem por objetivo principal facilitar o teste do sinal do resultado de cada soma ou subtração do laço principal do algoritmo de divisão.

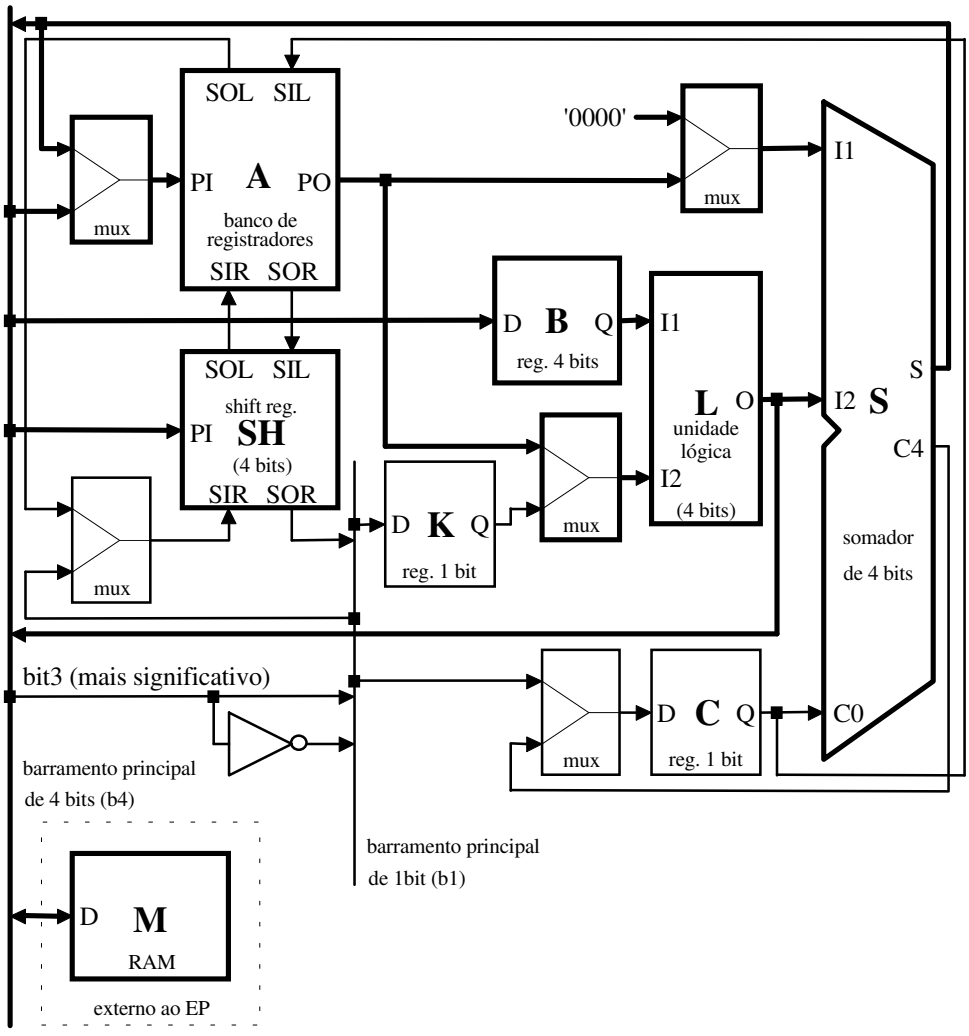

Figura III.21 - EP de 4 *bits* com banco de registradores para a divisão

Com relação à implementação do algoritmo de divisão, este processador se assemelha ao EP de 1 *bit* com fila FIFO, visto que a implementação depende do tamanho dos operandos. Sendo assim, foram desenvolvidos três algoritmos de divisão (Apêndice A), cada um com um limite diferente para o tamanho dos operandos. O primeiro algoritmo armazena a variável auxiliar x' inteiramente na memória interna, limitando o tamanho dos operandos a 16 *bits*. Já o segundo e terceiro algoritmos armazenam apenas a parte alta (mais significativa) da variável x' na memória interna. A parte baixa (menos significativa) é armazenada na área da memória externa reservada para o quociente *q* e, no caso do terceiro algoritmo, também no registrador de deslocamento (SH). O segundo algoritmo limita o tamanho dos operandos a 28 *bits*. Por fim, o terceiro algoritmo é específico para a divisão de inteiros representados por 32 *bits*.

O algoritmo desenvolvido para operandos limitados a 16 *bits* processa a divisão em $4(n/4)^2+10(n/4)+8$ ciclos. Inicialmente são gastos $n/4+1$ ciclos para o cálculo do módulo de *x*, $2n/4+1$ ciclos para o cálculo do módulo de *y* e um ciclo para atribuir um valor inicial aos registradores de máscara (K) e de *carry* (C). A primeira subtração demanda $n/4+1$ ciclos. O laço principal executa $n-1$ iterações. Cada iteração é composta por um deslocamento de 1 *bit*, uma soma ou subtração de *n bits* e duas atribuições de 1 *bit*, totalizando $n/4+1$ ciclos por iteração. Por fim, as correções finais (uma soma condicional de *n bits* e duas inversões de sinal condicionais de *n bits*) consomem $3n/4+5$ ciclos.

O algoritmo que executa a divisão usando um EP de 4 *bits* com banco de registradores para operandos limitados a 28 *bits* tem uma utilidade bastante limitada, visto que normalmente se trabalha com operandos que são potências de dois, podendo portanto utilizar o primeiro algoritmo apresentado, que é mais rápido. Este segundo algoritmo é um pouco mais lento. Ele processa a divisão em $4(n/4)^2+14(n/4)+4$ ciclos. Inicialmente são gastos $2n/4+1$ ciclos para o cálculo do módulo de *x* e $2n/4+1$ ciclos para o cálculo do módulo de *y*. A primeira subtração demanda $n/4+1$ ciclos. O laço principal executa $n-1$ iterações. Cada iteração é composta por um deslocamento de 1 *bit*, uma soma ou subtração de *n bits* e uma atribuição de 1 *bit*, totalizando $n/4+1$ ciclos por iteração. Além disto, a cada quatro iterações, são necessários mais dois acessos a memória, aumentando em $2n/4-2$ ciclos o custo do laço principal. Por fim, as correções finais (uma soma condicional de *n bits* e duas inversões de sinal condicionais de *n bits*) consomem $4n/4+4$ ciclos.

Por fim, o algoritmo desenvolvido para operandos de 32 *bits* processa a divisão em $4(n/4)^2+15(n/4)+3$ ciclos. Inicialmente são gastos $2n/4+1$ ciclos para o cálculo do módulo de *x* e $2n/4+1$ ciclos para o cálculo do módulo de *y*. A primeira subtração demanda $n/4+1$ ciclos. O laço principal executa $n-1$ iterações. Cada iteração é composta por um deslocamento de 1 *bit*, uma soma ou subtração de *n bits* e uma atribuição de 1 *bit*, totalizando $n/4+1$ ciclos por iteração. Além disto, a cada quatro iterações, são necessários mais dois

acessos a memória e um deslocamento de 1 *bit*, aumentando em $3n/4-3$ ciclos o custo do laço principal. Por fim, as correções finais (uma soma condicional de *n bits* e duas inversões de sinal condicionais de *n bits*) consomem $4n/4+4$ ciclos.

III.7 - Escolha do Elemento Processador

A Tabela III.2 apresenta uma comparação entre os Elementos Processadores propostos neste capítulo na execução das operações de soma, subtração, multiplicação e divisão de números inteiros representados em complemento a dois. Os desempenhos são apresentados em termos de número de ciclos de máquina para a execução de uma operação sobre operandos representados por *n bits*. Cinco diferentes Elementos Processadores são apresentados na tabela: EP1 (EP simples de 1 *bit*), EP1SH (EP de 1 *bit* com registrador de deslocamento programável), EP1FIFO (EP de 1 *bit* com fila FIFO), EP4 (EP simples de 4 *bits*) e EP4BR (EP de 4 *bits* com banco de registradores). Para facilitar a comparação, a Tabela III.3 apresenta os desempenhos (em número de ciclos) para três tamanhos de operando bem representativos: 8, 16 e 32 *bits*. Já a Figura III.22 apresenta uma comparação visual dos desempenhos das mesmas arquiteturas apenas para as operações sobre operandos de 16 *bits*.

Tabela III.2 - Ciclos necessários à execução de operações sobre números inteiros de n *bits*

	Soma/Subtração	Multiplicação	Divisão
EP1	$3n+1$	$3n^2+n$	$3n^2+10n+6$
EP1SH	$3n+1$	n^2+3n	$n^2+10n+6$
EP1FIFO	$3n+1$	n^2+3n	p / $n \le 16$: $n^2+7n+10$ p / $n \le 32$: $n^2+10n+7$
EP4	$3n/4+1$	$12(n/4)^2+10n/4+1$	$12(n/4)^2+25n/4+7$
EP4BR	$3n/4+1$	p / $n \le 28$: $4(n/4)^2+7n/4+2$ p / $n=32$: $4(n/4)^2+9n/4$	p / $n \le 16$: $4(n/4)^2+10n/4+8$ p / $n=32$: $4(n/4)^2+15n/4+3$

Tabela III.3 - Ciclos necessários à execução de operações sobre números inteiros

Operação	nº de *bits*	Ciclos Necessários à execução para cada EP				
		EP1	EP1SH	EP1FIFO	EP4	EP4BR
Soma/Subtração	8	25	25	25	7	7
	16	49	49	49	13	13
	32	97	97	97	25	25
Multiplicação	8	200	88	88	69	32
	16	776	304	304	233	94
	32	3080	1120	1120	849	328
Divisão	8	278	150	130	105	44
	16	934	422	378	299	112
	32	3398	1350	1351	975	379

Através das tabelas expostas, não é difícil concluir que o Elemento Processador de 4 *bits* com banco de registradores tem o melhor desempenho, dentre as arquiteturas apresentadas, para todas as operações analisadas. É bem verdade que o seu custo também é o mais elevado, porém a superioridade do seu desempenho compensa este aumento de custo. Observe que ele possui quase o mesmo número de registradores (memória interna) que o EP1SH ou que o EP1FIFO, porém ele chega a ser quase quatro vezes mais veloz do que estes processadores. Quanto a lógica necessária a um EP que trabalha com dados de 4 *bits*, ele chega a ser quase três vezes mais veloz do que o EP simples de 4 *bits* (EP4), que tem praticamente a mesma complexidade lógica. Sendo assim, o EP de 4 *bits* com banco de registradores foi considerado a melhor proposta de Elemento Processador para as operações aritméticas básicas sobre números inteiros. Uma análise mais detalhada da relação custo x desempenho das cinco propostas de arquitetura será apresentada no Capítulo VI.

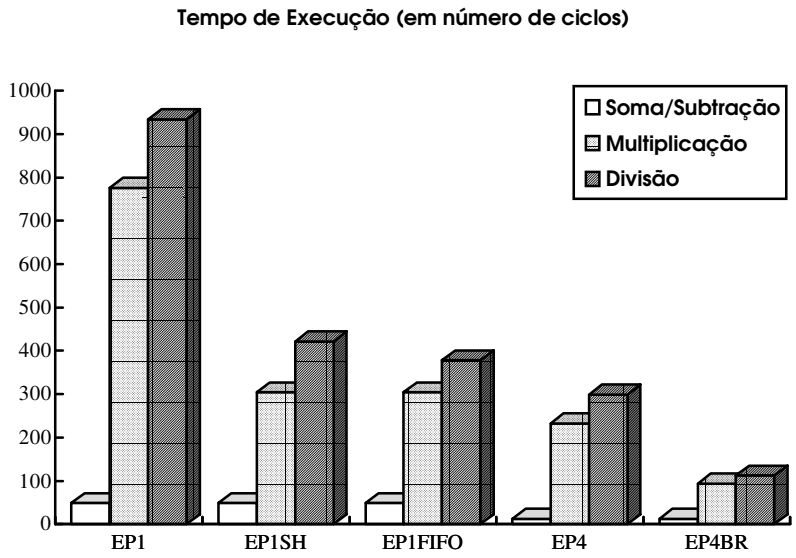

Figura III.22 - Ciclos necessários à execução de operações sobre números inteiros de 16 *bits*

Por último, a Tabela III.4 apresenta o desempenho de algumas arquiteturas comerciais vistas no Capítulo II na execução da soma, subtração e multiplicação de inteiros representados em complemento a dois por 8 *bits*. Esta tabela permite concluir que o EP de 4 *bits* com banco de registradores só perde em desempenho para o MasPar. Porém, o EP do MasPar tem um custo bem mais elevado, possuindo mais de 1.536 *bits* de memória interna, contra apenas 42 *bits* do EP4BR. Além disto, a sua lógica é bem mais complexa, incluindo unidades especiais de tratamento de expoente (com 16 *bits*) e de mantissa (com 64 *bits*) e unidade de cálculo de endereço (vide seção II.8). Deve-se considerar ainda que, os tempos apresentados para o MasPar supõem que um dos operandos está em um registrador especial e o resultado da operação será armazenado neste mesmo registrador especial. Além de constituir uma restrição

importante, este fator é preponderante para a superioridade de desempenho apresentada pelo MasPar, principalmente nas operações de soma e subtração.

Tabela III.4 - Ciclos necessários à execução de operações sobre números inteiros de 8 *bits*

Operação	DAP510	MPP	RPA	MasPar	EP4BR
Soma/Subtração	28	25	11	2	7
Multiplicação	250	88	80	25	32