Parte I - Projecto de Sistemas Digitais

Na disciplina de sistemas digitais foram estudadas técnicas de desenvolvimento de circuitos digitais ao nível da porta lógica, ou seja, os circuito digitais projectados, combinatórios ou sequenciais, são descritos como uma interligação de portas lógicas. Quando tentamos usar estas mesmas técnicas na concepção de circuitos de média e grande dimensão, rapidamente concluímos que se torna inviável devido ao elevado detalhe associado e à complexidade das técnicas que aumenta exponencialmente com o tamanho do problema.

Assim, necessariamente, o projecto de sistemas digitais de elevada complexidade não pode ser abordado ao nível da porta lógica, mas antes a um nível de detalhe que permita esconder pormenores de projecto e que reduza o número de elementos do projecto a um número que seja tratável pelo projectista. Esta forma de abordagem ao projecto de sistemas digitais que consiste em iniciar o projecto a um nível de abstracção elevado passando depois sucessivamente para níveis mais concretos com a concretização de determinado detalhes está já implícito quando se projecta com portas lógicas se tivermos em conta que estas são realizadas com transístores, um nível de abstracção mais baixo que o das portas lógicas.

O nível de abstracção que surge naturalmente a seguir ao da porta lógica é designado *nível de transferência de registos* (RTL – *Register Transfer Level*) e em que projecto se baseia na utilização de módulos mais complexos como *multiplexers*, contadores, operadores aritméticos, registos, memórias, etc. Se atentarmos à implementação de cada um destes módulos verificamos que consistem em várias porta lógicas, pelo que circuitos digitais com poucos módulos RTL podem conter centenas ou até milhares de portas lógicas cuja implementação ao nível da porta lógica seria muito trabalhoso senão impossível de concretizar em tempo razoável.

1.1 Descrição Funcional dos Módulos RTL

Os módulos usados na descrição de um circuito ao nível de abstracção RTL são em geral módulos de média complexidade que podem ser puramente combinatórios (e.g., *multiplexer*), puramente sequenciais (e.g., registo) ou um misto (e.g., contadores. São formados por um registo e um operador combinatório de incremento ou decremento). Sempre que necessário, o projectista pode incluir novos módulos RTL tendo que, nesse caso, indicar claramente qual a sua funcionalidade e como é implementado com portas lógicas ou com outros módulos RTL.

Os módulos RTL mais comuns são multiplexers, descodificadores, operadores lógicos, operadores aritméticos, registos, contadores, memórias. Em seguida descrevem-se alguns destes módulos, omitindo-se aqueles que foram estudados em sistemas digitais e para os quais não se pretende uma generalização (e.g., descodificador).

1.1.1 Multiplexer

O *multiplexer* é um módulo de encaminhamento de dados que selecciona de entre várias entradas uma cujos dados são enviados para a saída sem qualquer processamento intermédio. É assim formado por um conjunto de entradas de dados, uma entrada de selecção e uma saída. O valor na

entrada de selecção determina qual a entrada de dados a ser encaminhada para a saída. Genericamente, uma entrada de dados consiste em um ou mais bits, passando as entradas de selecção a serem comuns a todos os bits de uma entrada (ver figura 1).

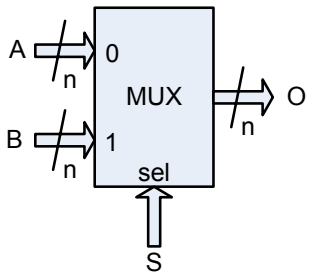


Figura 1 – Exemplo de um multiplexer com duas entradas de *n* bits cada

Em algumas implementações, um *multiplexer* pode conter um sinal de activação (*enable*). Quando activo, o *multiplexer* funciona como descrito anteriormente. Quando inactivo, o módulo coloca as saídas em alta impedância.

1.1.2 Operadores Lógicos

Os operadores lógicos (NOT, OR, AND, XOR) são usados ao nível lógico. Ao nível RTL, os operadores lógicos surgem geralmente agrupados num único módulo e recebem como entradas palavras, sendo a operação lógica aplicada bit a bit entre as palavra de entrada. Por exemplo, com A = '1010' e B = '1100' a operação lógica A OR B = '1110'. Este circuito seria implementado ao nível lógico com quatro portas OR. Genericamente, uma entrada de dados consiste em um ou mais bits, sendo a saída resultado da aplicação da operação lógica bit a bit (ver figura 2).

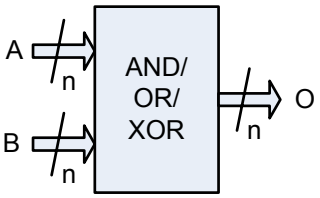


Figura 2 – Representação de um operador lógico binário com palavras de *n* bits

O operador inversão segue o mesmo raciocínio, mas com apenas um entrada, pois trata-se de um operador unário.

1.1.3 Operadores Aritméticos

Os operadores aritméticos são uma constante ao nível RTL. Os exemplos mais comuns são a soma/subtracção, multiplicação/divisão e raiz quadrada. No entanto, qualquer outro operador aritmético pode ser considerado, como por exemplo uma função transcendental, o logaritmo, a exponencial, etc.. Os módulos aritméticos têm geralmente apenas duas entradas de dados e uma saída com o resultado. A sua implementação com portas lógicas tanto pode ser puramente combinatória como sequencial, dependendo do algoritmo usado na sua implementação, bem como o desempenho pretendido. Genericamente, o módulo aritmético recebe duas entradas de *n* bits cada (excepto operadores unários. Por exemplo, a função $f(x) = x^2$, a função de incremento ou de decremento, etc.), sendo a saída resultado da aplicação da operação aritmética (ver figura 3).

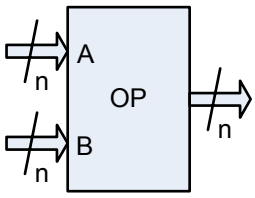


Figura 3 – Representação de um operador aritmético com duas entradas de palavras de *n* bits

1.1.4 Unidade Lógica Aritmética - ALU

Outros blocos aritméticos mais complexos podem ser feitos à custa dos blocos aritméticos básicos. Por exemplo, uma unidade lógica aritmética (ALU – *Arithmetic Logic Unit*) é um módulo capaz de executar mais de uma operação aritmética e/ou lógica. Neste caso, em que um módulo suporta a execução de mais de uma operação, é necessário uma entrada adicional que especifique a operação pretendida (ver figura 4).

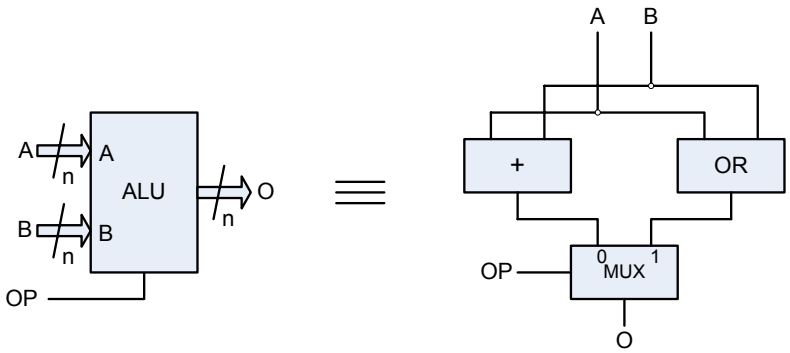


Figura 4 – ALU com duas operações

No exemplo da figura, temos a implementação de uma ALU com duas operações: soma aritmética e soma lógica. A saída é escolhida através de um *multiplexer* em que a entrada de selecção é controlada pela entrada OP da ALU que indica se pretende uma soma ou um OR.

1.1.5 Registos

Ao nível lógico estudou-se o flip-flop como um elemento que permite o armazenamento de um bit. Quando se pretende armazenar *n* bits então teremos de usar *n* flip-flops. Em muitos casos, pretende-se armazenar uma palavra de bits usada para representar um determinado valor ou informação. Neste caso, ao conjunto de flip-flops usados para armazenar a palavra designa-se *registo* (ver figura 5).

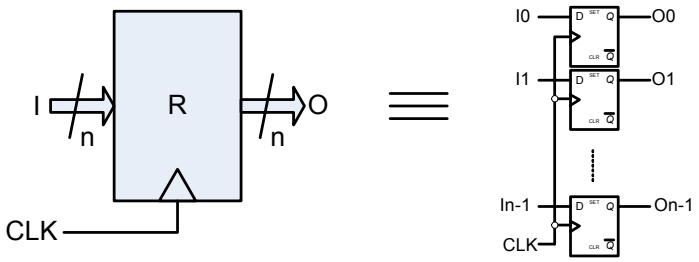


Figura 5 – Representação de um registo de palavras de *n* bits

O registo da figura é actualizado apenas no flanco ascendente do sinal de *clock*. Sempre que ocorre uma transição ascendente neste sinal, o registo guarda o valor presente à entrada até ao próximo flanco, ou seja, durante um ciclo de *clock*.

A funcionalidade de registo descrito está muito limitada pois o armazenamento da entrada é feito em todos os ciclos de *clock*. De forma a poder controlar em que momento se pretende registar a palavra de entrada usa-se um sinal de entrada designado *load*. Assim, durante a transição de *clock*, a entrada só é registada caso a entrada de *enable* esteja activa. Um outro sinal de controlo também bastante comum é o de *clear* ou *reset*, que quando coloca a 0 o conteúdo do registo. Este sinal pode ser síncrono ou assíncrono (a implementação de um registo com *load* e *reset* é deixada como exercício). Finalmente, um outro sinal com bastante utilidade prática é o que permite colocar as saídas do registo em alta impedância, designado *output enable* (OE). Na sua implementação, basta adicionar portas tri-state a cada uma das saídas do registo controladas pelo sinal OE.

Para além dos sinais de controlo do registo, podem-se facilmente adicionar saídas de estado, como a saída *zero* (Z) que indica se o conteúdo do registo é zero, negativo (N) que indica se o valor guardado é negativo, etc. A implementação destes bits de estado é simples. Por exemplo, a função Z é gerada à custa de uma NOR dos bits de saída (como implementaria a saída N?).

Para representar um registo com estes sinais basta incluir linhas de entrada e saída ao símbolo original (ver figura 6).

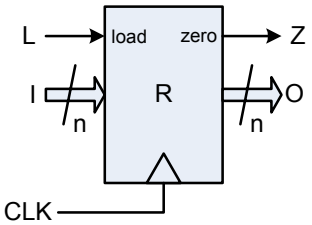


Figura 6 – registo com sinal de *load* e saída Z

No exemplo da figura temos um registo com sinal de controlo *load* e saída Z de estado.

1.1.6 Banco de Registos

Quando se pretende armazenar várias palavras utilizando vários registos pode-se optar por agrupar os registos gerando um módulo designado *banco de registos* (*register file*). Independentemente do número de registos do banco, em geral, a implementação do módulo não permite o acesso simultâneo a todos os registos. O mais comum é ter apenas uma entrada (um acesso de escrita) e uma ou mais saídas (leitura) (ver figura 7).

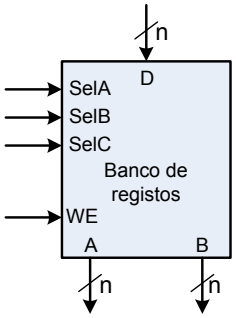


Figura 7 – Banco de registos com duas saídas e uma entrada

No exemplo da figura é possível apenas uma escrita de cada vez e duas leituras em simultâneo de qualquer um dos registos. A especificação dos registos destino (escrita) e origem (leitura) é feita através de sinais de controlo *Selx*. Adicionalmente, existe um sinal de *write enable* sobre a escrita. O módulo é controlado por um sinal de *clk*. Sempre que ocorrer uma transição de *clk* o registo destino seleccionado recebe a palavra de entrada caso o sinal de *write enable* esteja activo

Internamente, um banco de registos pode ser implementado com um bus único, com um bus de entrada e outro de saída ou então usando um descodificador à entrada e/ou *multiplexers* à saída (ver exemplo na figura 8).

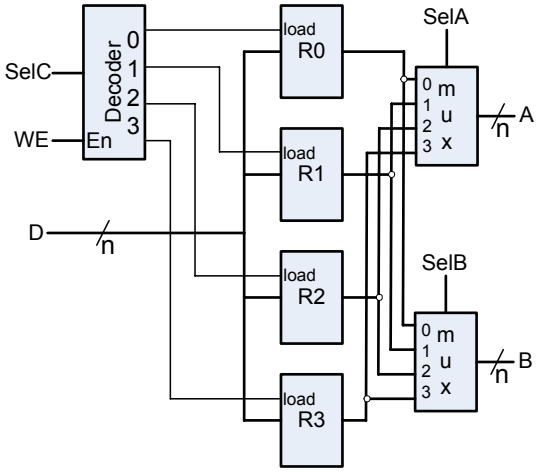


Figura 8 – Banco de registos implementado com um descodificador e *multiplexers*

Cada uma das saídas tem associado um *multiplexer* e a entrada tem um descodificador que activa o registo respectivo em função da entrada de selecção, *SelC*. O banco do exemplo tem quatro registos, pelo que cada sinal de selecção tem dois bits.

1.1.7 Memória RAM

.Quando se pretende armazenar um elevado conjunto de palavras recorre-se a um circuito de memória. Genericamente, um circuito de memória consegue armazenar *n* palavras com *m* bits cada (p×n). De entre os vários tipos de memória existentes, interessa-nos as memórias de acesso directo (RAM – *Random Access Memory*), que serão as únicas abordadas neste documento. Uma RAM tem um comportamento similar ao do banco de registos permitindo ao acesso a qualquer uma das posições de memória para escrita ou para leitura (ver representação na figura 9).

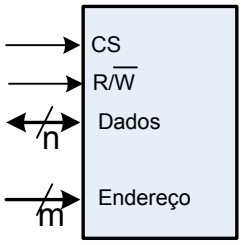


Figura 9 – Dispositivo de memória RAM

Uma RAM simples permite apenas uma operação de leitura ou de escrita em cada momento. Para tal, existe uma entrada de endereço para especificar a posição da memória que se pretende ler ou escrever e um bus de dados por são transportados os dados de ou para a memória. Repare que o bus de dados é bidireccional, por um lado porque em cada momento apenas é usado para entrada ou para saída, e por outro porque necessita de menos pinos no dispositivo. Adicionalmente, existem dois sinais de controlo. O sinal CS (*Chip Select*) controlo o estado de activação da memória. Quando activo permite operações de escrita ou de leitura. Quando inactivo, o bus de dados é colocado em alta impedância. O sinal de R/W é usado para indicar a operação (leitura ou escrita) pretendida. Os dispositivos RAM podem se assíncronos ou síncronos. No último caso, o dispositivo tem uma entrada adicional para o sinal de *clk*.

Para ler uma palavra do dispositivo RAM, basta colocar o endereço respectivo no bus de endereços e activar o sinal CS com R/W = 1. Para uma operação de escrita, altera-se o sinal R/W = 0, após os dados e os endereços estarem estáveis nos buses respectivos.

Internamente, um dispositivo de memória é formado por uma matriz de células de memória Cada célula de memória permite armazenar *n* bits.

1.2 Arquitectura de Sistemas Digitais de Elevada Complexidade

Os sistemas digitais de baixa complexidade são em geral abordados como uma única entidade devido à sua simplicidade. Muitos destes circuitos são exclusivamente de processamento ou de controlo, ou então circuitos de controlo com processamento reduzido. Em circuitos em que o número de variáveis e/ou de estados é elevado, é necessário estruturar o circuito por módulos. Uma das arquitecturas mais usadas na implementação destes circuitos consiste em duas unidades principais: unidade de processamento ou de caminho de dados (*datapath*) e unidade de controlo (ver figura 10).

A unidade de processamento é constituída por módulos de memória (ao nível RTL é formado por registos, memória RAM, etc.) e por módulos de processamento e de encaminhamento de dados (ao nível RTL é formado por *multiplexers*, contadores, operadores aritméticos). A unidade recebe dados do exterior, processa-os e envia os resultados para a saída de dados. As operações a serem realizadas pela unidade de processamento, bem como a sua sequência, são determinadas pela unidade de controlo através da palavra de controlo. A informação sobre o estado da unidade de processamento, de acordo com a sequência de operações realizada, é enviada à unidade de controlo na forma de uma palavra de estado. A interacção com o sistema é feita através de entradas e saídas de controlo.

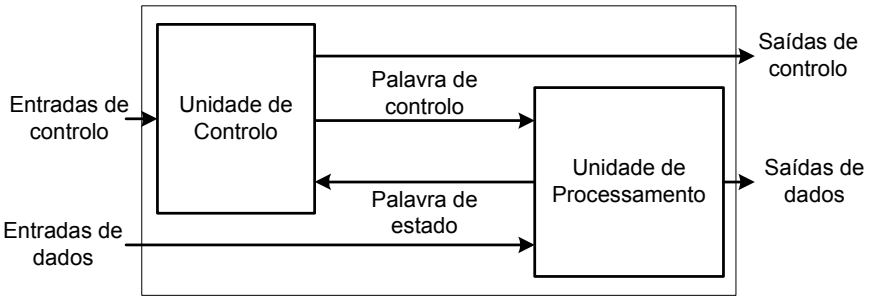


Figura 10 – Arquitectura de um circuito digital

De uma maneira geral podemos admitir o mesmo relógio para ambas as unidades, o que simplifica a sincronização entre os dois circuitos.

1.2.1 Projecto de um Circuito Multiplicador

Como exemplo de aplicação da arquitectura da figura 10, consideremos o projecto de um multiplicador P = A×B.

De uma forma trivial, poderíamos chegar facilmente a uma solução utilizando um módulo de multiplicação. No entanto, suponhamos que não temos esse bloco disponível, mas apenas um bloco aritmético de soma. Consideremos ainda que os parâmetros de entrada têm de ser armazenados antes de iniciar o cálculo. Existem dois sinais de início e de fim de operação.

Uma vez que apenas dispomos de um bloco de soma, podemos realizar a multiplicação à custa de somas sucessivas. Começamos por inicializar a zero um registo P e com o valor do multiplicador um registo M. Depois, enquanto M for diferente de zero, somamos ao registo P o valor do multiplicando (N) e decrementamos M.

A unidade de processamento necessita de três registos, P, M, N, para armazenar o produto, o multiplicador e o multiplicando, respectivamente. O registo P tem de incluir um sinal de *reset* e o registo M um sinal de *zero*. São ainda necessários um bloco de soma e um de decremento. O bloco soma recebe como operandos os valores de M e de P e gera um valor a ser armazenado em P. O bloco decremento recebe o valor de M e gera um valor a ser armazenado de novo em M (ver figura 11).

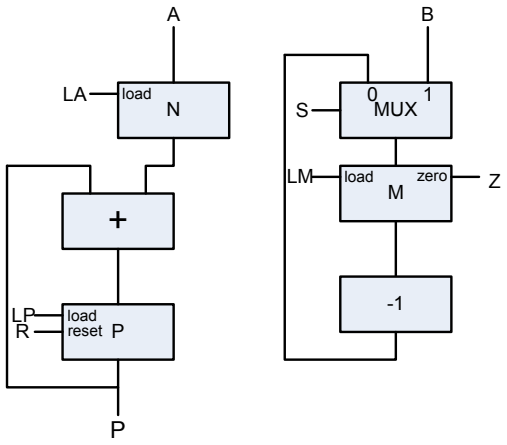


Figura 11 – Circuito de dados do multiplicador por somas sucessivas

O circuito de dados deve realizar as operações de acordo com a sequência especificada no algoritmo. Para descrever a sequência, vamos utilizar um fluxograma (ver figura 12a).

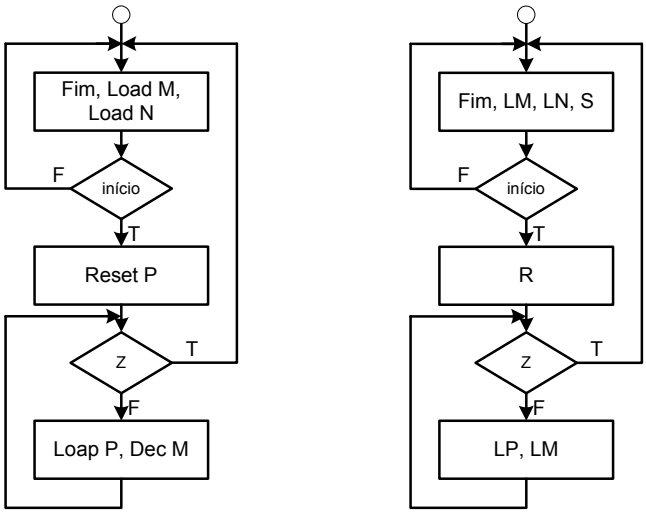


Figura 12 – Descrição do controlo do multiplicador

Após ter valores válidos nas entradas, activa-se o sinal *início* para dar início à operação de multiplicação. Após *reset* ao registo P, inicia-se o sinal de somas sucessivas até que o sinal de estado Z venha a zero. Nesta situação voltamos ao início da sequência e activa-se o sinal de *fim*.

Quando concretizamos as acções indicadas no fluxograma, passamos a ter os sinais de activação do circuito de dados (ver figura 12b). Inicialmente, activam-se os sinais LM, LN e o selector S do *multiplexer* para carregar os registos M e N. Após o sinal de *início*, faz-se *reset* a P através da entrada R. Durante o ciclo, apenas se têm de activar os sinais de *load*, LM e LP, dos registos M e P.

Quando juntamos os dois circuitos obtém-se o circuito ilustrado na figura 13.

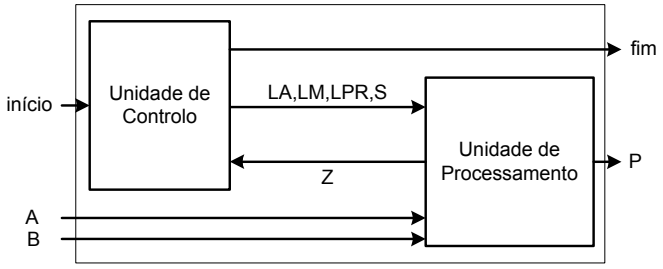


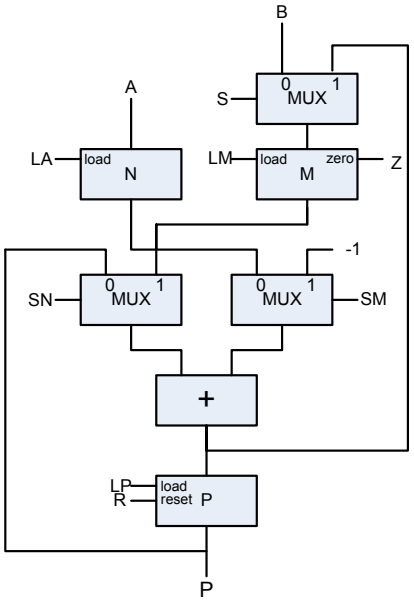
Figura 13 – Circuito completo do multiplicador

O exemplo ilustra bem a facilidade com que se chega a um circuito de relativa complexidade com base na arquitectura proposta e ao nível RTL.

A solução apresentada não é única, podendo-se ter optado por usar uma ALU em vez de dois módulos aritméticos separados ou até um único somador.

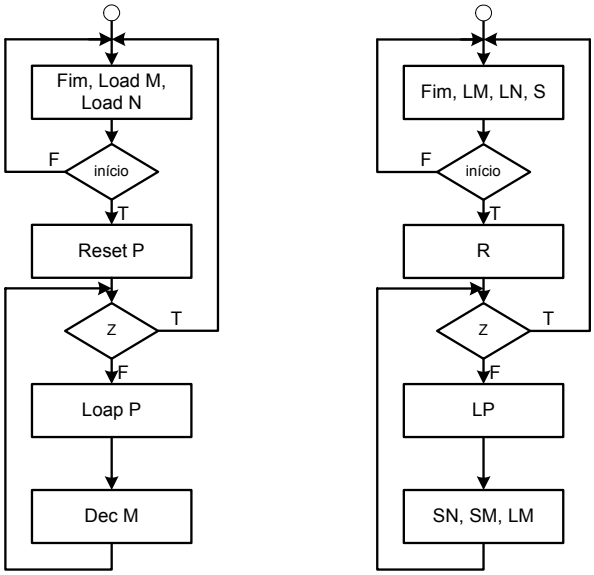
Exercício: Implemente o circuito de multiplicação por somas sucessivas usando apenas um somador.

Com apenas um somador, o decremento será feito à custa de uma soma com -1. As entradas do somador passam assim a ter de receber dois pares de valores (RP, RM) e (RN,-1) (ver figura).



Para poder seleccionar entre um par ou o outro foram adicionados *multiplexers* à entrada do somador. O resultado do somador será guardado no registo P, caso se trate da soma, e no registo M, caso se trate do decremento.

Consequentemente, o circuito de controlo terá de sequenciar as operações de soma e de decremento (ver figura).



A diferença relativamente à implementação anterior é que a última acção teve de ser sequencializada. Para carregar o registo P com a soma basta activar o *load* respectivo e seleccionar as entradas 0 de ambos os *multiplexers* de entrada do somador. Para carregar o registo M com o decremento é necessário activar o *load* respectivo e seleccionar as entradas 1 de ambos os *multiplexers* de entrada do somador.

Quando comparamos as soluções em termos de desempenho e de custo (em termos de área) verifica-se que a segunda solução é mais lenta, pois teve de serializar um par de operações. Quanto ao custo, a segunda solução temos menos um módulo de decremento, mas necessitou de mais dois multiplexers. A comparação depende da tecnologia usada na implementação do circuito.

1.2.2 Descrição de Hardware

O projecto de circuitos digitais começa com a sua especificação. Ao nível da porta lógica, a especificação é depois traduzida em funções lógicas e/ou em diagramas de estado. Quando o projecto se realiza ao nível RTL, a especificação terá de ser descrita necessariamente de outra forma. Uma das abordagens mais usadas usa as chamadas *linguagens de descrição de hardware* que permitem não só a descrição do circuito a vários níveis de abstracção como também a geração de uma descrição que pode ser usada como entrada das ferramentas de síntese de hardware.

Ao nível RTL as operações realizadas são basicamente de transferência entre registos, daí a sua designação. Neste sentido, apenas temos de considerar um formato de descrição que permita descrever as operações de transferência entre registos, bem como a sequência de operações de controlo. Neste sentido, vamos considerar um descrição baseada nas construções típicas das linguagens de programação (if .. then, while, for .. loop, etc.) e uma descrição adicional que representa a transferência entre registos com ou sem processamento de dados. Simbolicamente, teremos:

Rx ← Ry o registo Rx regista uma cópia de Ry

Rx ← Ry + Rz o registo Rx regista o valor da soma entre Ry e Rz

As operações a serem realizadas em simultâneo devem ser especificadas separadas por ';'. Por exemplo,

Rx ← Ry; R0 ← R1 + R2 A transferência de dados é realizada em simultâneo com a operação de soma.

Para exemplificar a utilização desta forma de descrição ao nível RTL, consideremos a descrição do circuito de multiplicação por somas sucessivas.

```
while (início = F) {
   fim ← T; RM ← A; RN ← B       //podem ser realizadas em paralelo
}
RP ← 0
while (RM ≠ 0) {
   RP ← RP + RN; RM ← RM - 1
}
```

Apesar de a especificação considerar a possibilidade de executar acções em paralelo, tal depende dos recursos da unidade de processamento. Por exemplo, Caso se considere que a unidade de processamento realiza as operações de soma e de decremento com um único somador, então as transferências RP ← RP + RN e RM ← RM – 1 não podem ser realizadas em paralelo.

Cabe ao projectista decidir que recursos usar em função dos requisitos de projecto (custo, desempenho, consumo de potência, etc.).

1.3 Exercícios

Exercício 1: Implemente um circuito de divisão de inteiros por subtracções sucessivas.

Exercício 2: Implemente um circuito que determine o máximo divisor comum entre dois números. Para tal, considere o algoritmo seguinte:

```
Maior_Divisor_Comum(X, Y)
1. while (Y ≠ 0){
2.   if X ≥Y
3.     X=X-Y
4.   else
5.     troca X com Y
6.}
```

Por palavras, neste algoritmo subtrai-se sucessivamente o menor dos números ao maior até que o resultado desta subtracção seja 0. Quando isso acontece, o algoritmo termina e o resultado é o valor final do outro operando. Não é um algoritmo muito eficiente pois, por vezes, demora muito a terminar, mas é simples de realizar.

Assume-se que na especificação do sistema se indica que os operandos X e Y se encontram inicialmente guardados em dois registos, designados por Rx e Ry. Além disso, essa especificação indica que existem dois sinais de controlo, um sinal de entrada *início* para indicar que os registos Rx e Ry foram carregados com os operandos e que se deve dar início ao cálculo do maior divisor comum entre eles, um sinal de saída *fim* que assinala o fim deste cálculo.

Exercício 3: Implemente o circuito de multiplicação por somas sucessivas utilizando um banco de registos e um somador.