SISTEMAS ELECTRÓNICOS DE COMPUTADORES

Leonel Augusto Pires Seabra de Sousa

Francisco André Corrêa Alegria

14 de setembro de 2015

CONTEÚDO

1. INTRODUÇÃO

1.1 Electrónica de Computadores

Os computadores estão em todo o lado. Por computador entende-se não só o que no dia-a-dia chamamos de computador, como o computador de uso geral portátil ou de secretária, mas todos os dispositivos electrónicos que contêm uma unidade de processamento de dados ou informação, como seja um microprocessador ou microcontrolador. A utilização dessa classe de dispositivos estende-se hoje à electrónica de consumo, desde telemóveis e televisões a frigoríficos e automóveis.

Uma das formas de definir essas duas classes de computadores é

- **Sistemas Computacionais Embebidos** – Têm como objectivo *servir uma aplicação* como, por exemplo, um leitor de DVD ou uma máquina de fax. A sua existência passa despercebida ao utilizador. O que para ele interessa é a aplicação em si.
- **Computadores Genéricos** – Têm como objectivo *servir o utilizador*, sendo este quem define a aplicação. Esta pode ser o processamento de texto, o envio de uma mensagem de correio electrónico, um jogo ou o alojamento de uma página na Internet, por exemplo.

Os sistemas embebidos têm uma funcionalidade simples executando repetidamente um único programa. As suas especificações são apertadas visando o baixo custo e o reduzido consumo energético. Espera-se de um sistema embebido que seja reactivo ao utilizador e ao meio envolvente e que opere em tempo real realizando os seus cálculos sem atraso aparente.

A Figura 1.1 apresenta o diagrama de blocos típico de uma câmara fotográfica digital. Observe-se que neste caso existe um microcontrolador encarregue da interface com o utilizador (botões, visor, etc.) e do controlo do diafragma, entre outras coisas, e um processador encarregue do processamento e armazenamento de imagem bem como da comunicação com o exterior (através da interface USB, por exemplo). O processador é também responsável pela produção dos avisos sonoros ao utilizador e por processar o som capturado com um microfone.

Este sistema electrónico embebido tem como única função tirar e armazenar fotografias (eventualmente associadas a som para fácil catalogação). As maiores preocupações no projecto de um sistema deste tipo é que seja pequeno e leve de modo a ser facilmente transportável, que tenha um baixo consumo de modo a estender ao máximo o tempo de vida das baterias usadas para o alimentar, que funcione de forma rápida possibilitando a recolha de várias fotografias em sucessão rápida e que tenha uma interface com o utilizador reactiva de modo a proporcionar uma boa experiência de utilização.

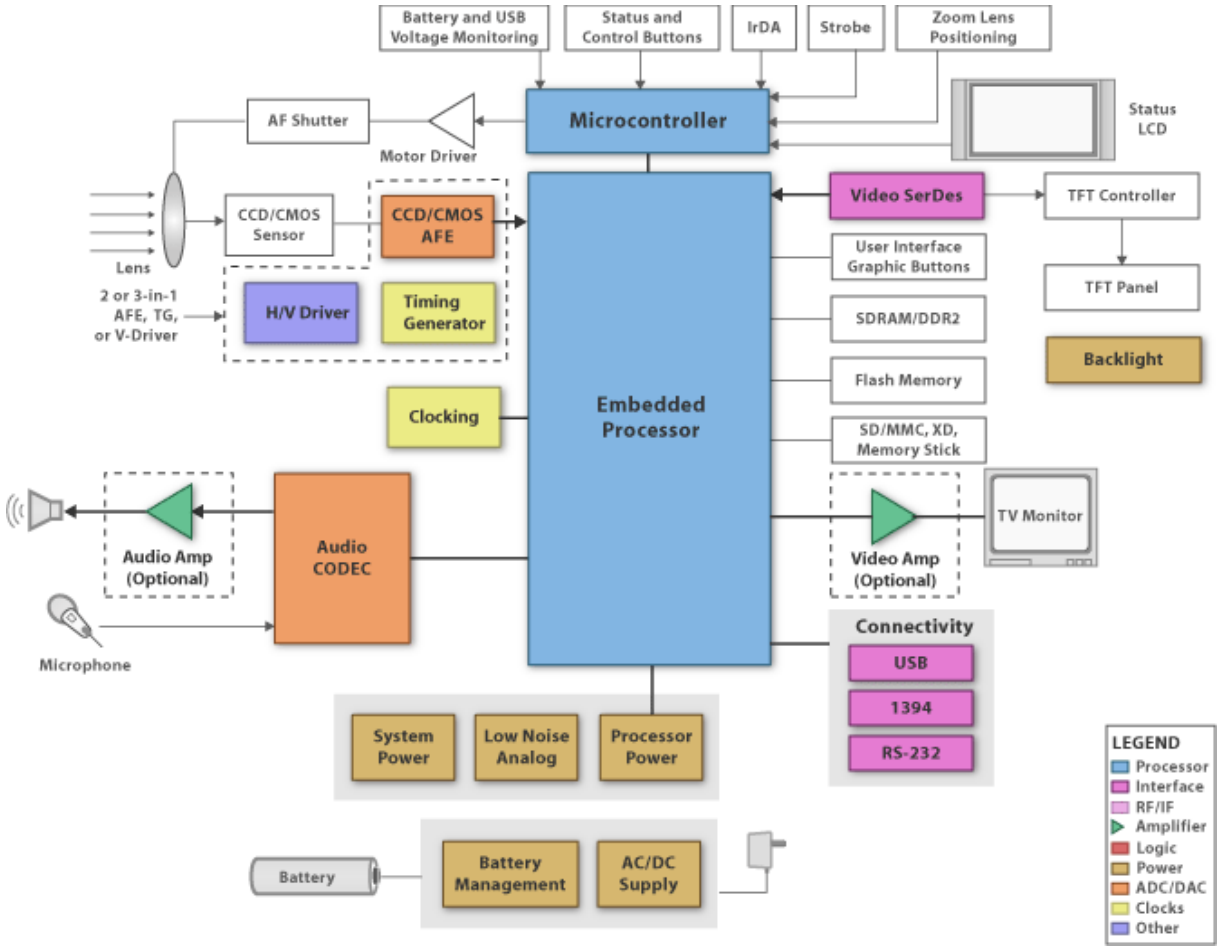


Figura 1.1 – Diagrama de blocos de uma câmara fotográfica digital. Copyright Texas Instruments. Retirada de http://focus.ti.com/docs/solution/folders/print/80.html.

A importância primordial que os computadores têm hoje em dia na vida de cada um explica-se de forma resumida numa única frase – *Os computadores tornam a nossa vida mais fácil*. Muitas das actividades do Homem podem ser realizadas, de forma mais rápida e melhor, com o auxílio de computadores, como seja o editar de um livro ou a comunicação à distância. Por outro lado, usando computadores, podemos fazer coisas que de outro modo seriam praticamente impossíveis como, por exemplo, evitar a colisão entre automóveis em piso molhado com o sistema ABS (*Anti-lock Braking System*[1]).

Não há dúvida que a produtividade do Homem aumentou consideravelmente com o advento dos computadores. No entanto, ao contrário do que seria de esperar, o tempo de lazer não aumentou na mesma proporção. Na verdade até diminuiu, embora isso não seja devido unicamente ao computador. Seja como for, continuamos a querer ter computadores que façam mais coisas e sejam mais rápidos. No fundo gostaríamos que o computador fosse de facto inteligente, não só para facilitar

[1] Originalmente o acrónimo ABS advinha da expressão alemã *Antiblockier-Bremssystem* dado que foi criado pela empresa alemã Bosch (em 1978).

a interacção com "ele" mas também para que fosse capaz de fazer muitas das coisas que temos que fazer e não gostamos. No dia em que aí chegarmos com certeza arranjaremos outros trabalhos e obrigações para nos mantermos ocupados.

1.2 Projecto de Sistemas Electrónicos

1.2.1 Gestão do Projecto

A gestão de um projecto de electrónica é uma tarefa complicada, especialmente em produtos complexos. É, no entanto, bastante importante para o sucesso de um produto. Essa gestão, que deve naturalmente estar a cargo de pessoas com experiência nessa área, deve cobrir diferentes aspectos como a constituição da equipe de desenvolvimento, a divisão do desenvolvimento em partes, a escolha de tecnologias de desenvolvimento, o fornecimento de componentes, a implementação de procedimentos de validação e verificação do desenvolvimento adequados, a planificação da produção em massa do produto e a criação de uma estratégia e campanha de marketing.

Uma das primeiras tarefas consiste em dividir o desenvolvimento em diferentes partes tão independentes quanto possível, de modo a que o seu desenvolvimento possa ocorrem em paralelo, tornando-o mais rápido. Essa divisão pode ser feita de forma hierárquica chegando, no limite, à divisão em módulos com uma única função. Um tipo de divisão em geral efectuado consiste na separação do desenvolvimento do hardware e do software. Outra escolha consiste na decisão do uso de módulos de software e hardware já existentes no mercado ou desenvolvidos pela empresa em projectos anteriores.

Essa divisão do desenvolvimento em partes é seguida pela criação de equipas encarregues das diferentes partes. Essas equipas são criadas com base na área de especialidade de cada engenheiro tendo em conta as suas relações inter-pessoais e nível de experiência – não devem estar na mesma equipa pessoas que não se dão bem nem constituir uma equipa só com pessoas que tenham pouca experiência.

Cada equipa deve ter um líder, uma tarefa clara e prazos bem definidos. Cada membro da equipa deve ter uma responsabilidade atribuída e deve ser avaliado continuamente para detecção de dificuldades, problemas e ineficiências.

Há que prestar especial atenção à comunicação entre equipas de modo a que o trabalho de cada uma possa facilmente ser integrado num único produto numa fase final do projecto.

O desenvolvimento de cada uma dessas partes e do produto completo inicia-se na concepção, seguida da implementação e construção de protótipos, e finalmente do ensaio. Nesse ponto pode-se decidir voltar, quantas vezes forem necessárias, à fase de projecto e à construção de protótipos.

1.2.2 Métricas de Avaliação de Projecto

A criação de um computador/sistema embebido para determinada aplicação é um processo que não tem uma única solução. Existem vários factores que influenciam esse desenvolvimento e que

acabam por determinar o produto final obtido. Esses factores concorrem entre si, sendo muitas vezes necessário fazer compromissos de acordo com os constrangimentos impostos.

Existem diferentes métricas usadas para caracterizar a implementação de um sistema:

- **Custo NRE** – O custo NRE (Non-recurring engineering) representa o custo do desenvolvimento do produto e é independente do número de unidades produzidas. Depois de lançado o produto este custo deixa de existir. Quanto mais complexo o produto desenvolvido, mais tempo é preciso para o desenvolver e maior é o número de pessoas envolvidas, aumentando o custo. Obviamente quanto menor o custo NRE melhor. Algumas das formas de conseguir isso são: i) usar uma equipa de desenvolvimento que já tenha experiência em desenvolver produtos semelhantes; ii) organizar o desenvolvimento em equipas bem dimensionadas e que usem formas eficientes de comunicação entre elas de forma a eliminar a sobreposição de trabalho e focar o esforço de cada membro da equipa tendo em conta a sua área de especialidade; iii) adquirir máquinas e ferramentas apropriadas para aumentar a eficiência do trabalho desenvolvido; e iv) optar por usar componentes existentes no mercado (COTS – *Commercial Off-The-Shelf*) em vez de os desenvolver de raiz. Isso poupa tempo e recursos no desenvolvimento e na manutenção.
- **Custo Unitário** – O custo de produzir uma única unidade do produto. Este custo exclui o custo NRE. Quanto menor o custo unitário, maior pode ser a margem de lucro e/ou menor pode ser o preço final tornando o produto mais competitivo. Este custo reduz-se optando, durante o desenvolvimento, pelo uso de componentes mais baratos. Pode ser justificado o desenvolvimento de raiz de alguns componentes em vez de usar COTS quando este é muito específico ou não exista no mercado um produto com as funções desejadas ou com funções a mais que não são precisas. Em situações típicas, no entanto, o uso de COTS torna mais barato o produto final devido ao grande número de unidade dos COTS vendidas. Outra forma de reduzir o custo unitário consiste em produzir o produto em série e optimizar os processos de produção de modo a diminuir a quantidade de mão-de-obra necessária, usando máquinas e automatizando processos.
- **Custo Total** – O custo total da criação de um produto comercial tem portanto duas componentes: o custo de desenvolvimento e o custo de produção. Este último, por sua vez, é igual ao custo unitário multiplicado pelo número de unidades produzidas.
- **Custo Real** – O custo real de uma unidade produzida é a soma do custo de produção de unidade (custo unitário) com o custo de desenvolvimento (NRE) dividido pelo número de unidades produzidas. Por exemplo, no caso de um produto que custou 200 000 € a desenvolver e do qual se produziram 1000 unidades com um custo unitário de 50 €, o custo real de cada unidade é de 250 € (200 000/1000 + 50). A amortização do custo de desenvolvimento é portanto de 200 € por unidade. Quanto mais unidades forem produzidas mais o custo real se aproxima do custo unitário.
- **Tamanho** – Quantifica o volume ocupado pelo produto acabado, o número de transístores usados, no caso de um circuito integrado, ou o número de linhas de código ou bytes ocupados no caso de software. O tamanho do produto acabado assume maior importância, a par com o peso, no caso de produtos que se pretende sejam portáteis como telemóveis e

computadores portáteis, por exemplo. No caso de circuitos electrónicos o número de transístores determina a área ocupada num circuito integrado. Quanto menor a área, menor o custo unitário. Por outro lado, na fase de desenvolvimento, diferentes circuitos integrados com pequena área podem ser incluídos no mesmo protótipo de circuito integrado baixando o custo NRE. No caso de software, o número de bytes é importante pois determina o tamanho da memória que é necessária incluir no produto para armazenar o programa e consequentemente o custo unitário dessa memória. O número de linhas de código é importante porque por um lado determina o custo do desenvolvimento, e por outro, o custo de manutenção.

- **Desempenho** – É quantificado com o tempo de execução ou o ritmo de dados (transmitidos ou processados). Um maior desempenho em determinada área permite uma maior competitividade em relação a produtos semelhantes existentes no mercado, e portanto um maior número de vendas. Naturalmente que para obter um maior desempenho é muitas vezes necessário componentes mais caros o que aumenta o custo unitário do produto. Por outro lado, um maior desempenho leva muitas vezes a um maior consumo energético o que pode ser indesejado em produtos alimentados por baterias como sensores, telemóveis ou computadores portáteis. O objectivo último, que todas as equipas de desenvolvimento procuram alcançar, é o de ter uma ideia inovadora e engenhosa capaz de levar à construção de um circuito mais rápido, com menos consumo e mais barato.
- **Consumo Energético** – O consumo energético está intimamente relacionado com o tempo de vida da bateria. Por outro lado um maior consumo energético leva normalmente a uma maior dissipação de energia e consequente aumento da temperatura, o que leva a uma maior necessidade de arrefecimento. Isso, por sua vez, em geral leva a um maior custo unitário e a um maior tamanho do produto final.
- **Flexibilidade** – A flexibilidade mede quão fácil é mudar ou acrescentar as funções de um produto. Isso trás vantagens em termos competitivos pois permite reagir de forma rápida a desenvolvimentos efectuados por empresas concorrentes. Em produtos cujo tempo de desenvolvimento é longo (alguns anos), muitas vezes acontece que durante esse desenvolvimento surgem no mercado produtos concorrentes com novas funções que não tinham sido previstas no inicio do desenvolvimento. Para que o produto lançado seja competitivo é muitas vezes necessário altera-lo de modo a incluir essas funções disponibilizadas pela concorrência. Um produto flexível permite faze-lo com o menor custo e atraso possível. O software é muito mais flexível do que o hardware. Esta é portanto uma das razões que faz as equipas de desenvolvimento optarem por implementarem certa funcionalidade em software quando existe a possibilidade de o fazerem em hardware, muitas vezes até com benefícios em termos de desempenho.
- ***Time-to-prototype*** – Tempo que demora o desenvolvimento de um produto até ser construído o primeiro protótipo. Esse marco é importante pois valida as escolhas efectuadas e demonstra que o produto projectado de facto tem o desempenho apropriado. Muitas vezes o custo dos protótipos é maior do que o custo unitário do produto pois é criado de forma muitas vezes manual e em pequeno número. Um curto tempo de desenvolvimento até ao primeiro protótipo é essencial pois dá mais liberdade para redesenhar o produto caso seja necessário,

leva a um tempo de entrada em mercado mais curto e permite satisfazer as exigências dos investidores existentes e angariar novos investidores.

- ***Time-to-market*** – Tempo necessário para desenhar e produzir o produto de forma a estar pronto para venda. Um curto tempo de chegada ao mercado pode trazer enormes lucros devido a uma menor concorrência que existe quando um produto é novo e que entretanto vai aumentado quando as empresas concorrentes desenvolverem produtos semelhantes. O atraso de alguns meses apenas pode ter um grande impacto no lucro obtido. Hoje em dia este tempo é em média de 8 meses.
- ***Janela de mercado*** – É o intervalo de tempo em que um produto tem a mais alta taxa de vendas (Figura 1.2). Ter um produto no mercado durante esse intervalo muitas vezes impõe um período de desenvolvimento curto.

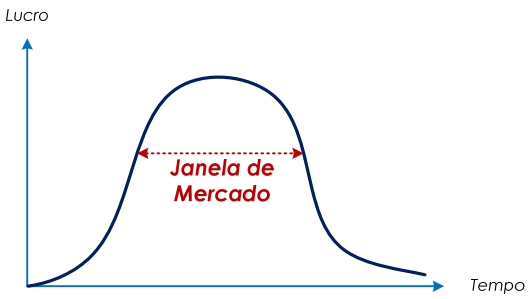


Figura 1.2 – Ilustração da janela de mercado.

- **Segurança** – Mede a probabilidade que o produto desenvolvido tem de funcionar de forma incorrecta e provocar ferimentos ao utilizador. Se isso acontecer, além do custo de indemnização incorre-se em enormes custos em termos de marketing e de aceitação do produto por parte dos potenciais compradores.
- **Preocupação ambiental** – O uso eficiente de energia e a não utilização de materiais nocivos para o ambiente é um requisito exigido por certos países e um bom argumento de marketing que pode contribuir para o sucesso de um produto.

Um produto pequeno, seguro, amigo do ambiente, com um óptimo desempenho, baixo consumo, baixo custo unitário e baixo NRE e desenvolvido em poucos meses seria o ideal. Na prática há que fazer compromissos entre estas diferentes métricas de forma a criar um produto funcional com boa aceitação no mercado e que origine lucro.

Na escolha do tipo de processador, por exemplo, várias destas métricas têm que ser consideradas sendo o equilíbrio ideal dependente da aplicação. A escolha mais acertada na maioria dos casos será a de um processador de uso geral existente no mercado. A principal vantagem é o reduzido custo NRE e tempo de desenvolvimento já que não é preciso desenvolver o hardware do processador. Só o software. Outras vantagens passam pela flexibilidade já que facilmente podem ser alteradas e acrescentadas funcionalidades ao produto, e pelo reduzido custo unitário já que o mercado desses processadores é muito grande e portanto o fabricante amortiza os custos de desenvolvimento por um

grande número de unidades. Por outro lado esses processadores são bem optimizados dando origem a um bom desempenho em aplicações onde são realizados muitos cálculos básicos.

As desvantagens que esta solução trás têm a ver com o desempenho em certas aplicações, que pode ser baixo, e com o tamanho e consumo energético já que esses processadores são projectados para um conjunto genérico de aplicações e portanto têm que prever funcionalidades e capacidades que podem não ser necessários numa dada aplicação.

Em certas casos estas desvantagens podem ser minoradas pela escolha de um processador de uso geral especializado para um dado tipo de aplicação (ASIP – *application-specific processor*) como o processamento de sinal ou o controlo. O facto de existirem processadores desses no mercado faz com que as vantagens em termos de custo de desenvolvimento sejam as mesmas que no caso do uso de um processador de uso geral. Por outro lado, como são específicos para esses tipos de aplicações, conseguem ser mais rápidos.

Os microcontroladores, por exemplo, são um exemplo de ASIP vocacionados para o controlo. Possuem assim entradas e saídas analógicas e digitais bem como interfaces do tipo RS232, USB, I^2C, SPI e CAN. O seu uso evita portanto que se tenha que criar o hardware necessário para implementar essas funções, como conversores analógicos/digitais, que não existem em processadores de uso geral. Naturalmente que numa aplicação em que só fosse preciso, por exemplo, a interface USB e as entradas digitais, ganharia em termos de consumo e tamanho se fosse projectado um processador especializado que só tivesse mesmo essas interfaces. Na maioria dos casos, no entanto, o custo muito maior de desenvolvimento não justifica a escolha dessa opção.

A escolha pelo desenvolvimento de um processador de raiz especializado para realizar a função pretendida pode no entanto ser a mais acertada em alguns casos, especialmente nos casos em que essa função seja bem definida, não susceptível de ser alterada no futuro e que o mercado para o produto seja suficientemente grande de forma a ser possível amortizar o custo de desenvolvimento, naturalmente mais alto do que no caso do uso de um processador de uso geral existente no mercado, pelas unidades produzidas. Também o desempenho pode ser melhor e o tamanho e consumo menores.

1.2.3 Tecnologias de Projecto

O desenvolvimento de um produto hoje em dia é feita de cima para baixo, isto, é, começando de um nível de abstracção superior e refinando-se as soluções até à implementação concreta dos circuitos electrónicos ou do software. No nível superior o sistema é especificado usado linguagem natural ou uma linguagem do tipo C. A especificação é então refinada de modo a distribui-la por processadores de uso geral ou dedicados dando origem a especificações comportamentais. Essas, por sua vez, são refinadas em código *assembly* no caso de processadores de uso genérico, ou em componentes de transferência entre registos e máquinas de estado no caso de processadores dedicados. Essa especificação, por sua vez, é refinada em termos de equações booleanas. Finalmente chega-se à implementação através de código máquina em processadores de uso geral e através de circuitos lógicos digitais no caso de processadores dedicados (Figura 1.3).

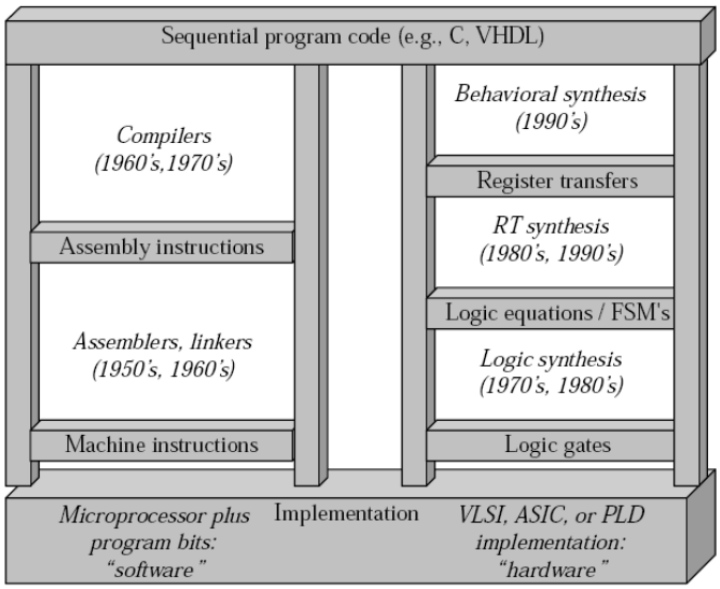


Figura 1.3 – Ilustração dos diferentes refinamentos efectuados nas componentes de software e hardware.

Um dos avanços recentes em termos de desenvolvimento consiste no uso da compilação, no caso do software e na síntese, no caso do hardware, para refinar uma especificação desde o nível mais alto. Isso aumenta a produtividade pois o engenheiro concentra a o seu esforço na especificação de alto nível e o resto é criado de forma automática.

Outra tecnologia usada para aumentar a produtividade é usar, no caso de software, bibliotecas de funções já existentes. No caso de hardware tal também é possível fazer, sendo designando-se por "propriedade intelectual" o uso de circuitos electrónicos desenvolvidos e vendidos por outros.

1.2.4 Especificação do Produto

Uma das primeiras actividades realizadas no desenvolvimento de um novo produto é a compilação de um conjunto de especificações que este deve satisfazer e funcionalidades que deve possuir. Isso é, em geral, feito através de entrevistas ao cliente ou à pessoa que propôs o desenvolvimento do produto em causa. Na prática os clientes querem sempre tudo, para ontem e sem custos. A melhor abordagem é não perguntar o que querem que o produto faça mas sim qual a sua função e quem o usará. A partir daí o engenheiro será capaz de criar uma lista de especificações e funcionalidades mais realista.

Essas especificações devem ser ordenadas por grau de importância, cobrir todas as facetas do produto, não devendo haver conflitos entre elas nem ambiguidade na sua interpretação.

1.2.5 Escolha dos componentes

Na escolha do tipo de hardware a utilizar pode-se considerar o desenvolvimento de hardware específico ou a utilização de hardware já existente. A escolha nem sempre é evidente. Um circuito digital desenvolvido de propósito para a uma dada aplicação nem sempre é mais rápido a executar as funções pretendidas do que um microprocessador de uso geral. Isso acontece porque os

microprocessadores são desenvolvidos por grandes equipas que têm tempo e recursos para optimizar o funcionamento do microprocessador tanto em termos de rapidez como de consumo.

Por outro lado, um dispositivo que necessite de executar várias funções teria que ter vários circuitos diferentes para cada uma dessas funções os quais passariam a maior parte do tempo se ser usados. O uso de um microprocessador, por outro lado, permite que o mesmo circuito (o contido no microprocessador) seja usado para as várias funções. Obtém-se assim um ganho em termos de tamanho e custo do hardware.

Uma pequena equipa sujeita a prazos apertados dificilmente consegue criar, ensaiar e optimizar um circuito de forma a extrair todo o desempenho que em princípio está disponível pela costumização a uma aplicação específica. Essa solução, em geral, só vale a pena para equipas grandes e quando o número de unidades produzidas seja grande, de forma a amortizar os custos do desenvolvimento.

O uso de um microprocessador permite também que o desenvolvimento do produto possa ser separado na vertente de hardware e software.

Para além dos circuitos lógicos que efectuam a computação propriamente dita (microprocessador ou circuito lógico construído de propósito), são necessários vários outros componentes como memória e periféricos para interface com o utilizador ou outros sistemas. Há que pesar bem quanto hardware é necessário. Se forem usados componentes a menos, com o objectivo de reduzir o tamanho ou o custo, por exemplo, pode acontecer que o produto desenvolvido não tenha todas as funcionalidades desejadas. Por outro lado, o uso de componentes a mais, como, por exemplo, o uso de memórias com muita capacidade, leva a que o custo seja superior ao necessário o que pode condicionar fortemente o sucesso de um produto no mercado especialmente tendo em conta a competição existente.

Uma das condicionantes ao desenvolvimento de uma determinada solução é, hoje em dia, o consumo energético. Numa altura em que cada vez mais se procuram produtos leves e portáteis a alimentação por bateria ou painéis solares é cada vez mais comum. Isto leva a que o sistema electrónico tenha que consumir o menos possível de modo a prolongar o período de funcionamento entre recargas. Em certos casos os produtos desenvolvidos devem ser instalados em locais remotos de difícil acesso em que se torna impraticável o recarregamento frequente das baterias. Há portanto que escolher componentes de baixo consumo, implementar um procedimento de desligar ou adormecer partes do circuito quando não estão a ser usadas e escolher a frequência de relógio o mais baixo possível sem comprometer o desempenho do dispositivo. Muitas vezes usar a maior frequência disponível num dado processador não faz sentido já que o desempenho é limitado por outros factores como o tempo de acesso à memória ou a necessidade de esperar pela introdução de dados por parte do utilizador.

Existem directivas da União Europeia que impõem certas restrições aos materiais utilizados de modo a minimizar o impacto que os produtos electrónicos têm no ambiente. A directiva 2002/95/EC, conhecida como directiva RoHS (*Restriction of Certain Hazardous Substances*) proíbe o uso de Cádmio, Mercúrio, Crómio Hexavalente, Bifenilos Polibromados, Éteres Difenil-polibromados e Chumbo. A directiva 2002/96/EC, conhecida por WEEE (*Waste from Electrical and Electronic Equipment*), por

outro lado, atribui aos fabricantes a responsabilidade pela reutilização ou destruição de forma ecologia de produtos eléctricos e electrónicos descartados.

1.2.6 Interface com o Utilizador

Há muitos computadores embebidos que passam despercebidos ao utilizador do sistema. Isso acontece muitas vezes, por exemplo, nos automóveis. Há no entanto muitos outros sistemas contendo computadores em que uma das principais funções é a interface com o utilizador. Nesses casos é preciso prestar especial atenção ao seu desenho e construção.

Do ponto de vista do engenheiro envolvido na criação de um produto e que tem que desenvolver o hardware ou software, a interface com o utilizador é encarada como sendo o sistema que permite que a informação e os comandos cheguem ao computador e que, por outro lado, é o destino da informação resultante do processamento executado. Esse ponto de vista é no entanto demasiado estrito resultando muitas vezes em produtos, que até podem ser muito bons em termos de funcionamento, mas que têm uma má interface com o utilizador. Como é sabido, especialmente hoje em dia, a aparência e a percepção que as pessoas têm dos produtos é tão ou mais importante que o produto em si. É ela no fundo que dita as vendas e o sucesso de um produto.

O desenvolvimento da interface com o utilizador, tanto na vertente de hardware como de software, deve ser encarada do ponto de vista do utilizador e não do ponto de vista do engenheiro que desenvolve o produto. Não deve ser esquecido que o produto desenvolvido é criado para servir o utilizador. Alguns dos aspectos a ter em atenção são, por exemplo:

- **Expectativa do Utilizador** – Ao desenhar a interface com o utilizador para um dado produto deve-se ter em conta o que o utilizador do produto espera deste. Por exemplo, se um utilizador adquire um leitor de DVD, ele espera que haja um tabuleiro ou ranhura onde colocar o DVD. Essa operação será por ventura a primeira realizada quando começa a usar um produto novo. Ao desenhar-se o leitor de DVD deve-se portanto tornar claro para o utilizador onde deve ser introduzido o DVD. Isso passa por não colocar essa entrada na parte de trás ou de lado do leitor. Os utilizadores esperam que o DVD seja colocado na parte da frente. É aí que isso deve acontecer, estando, para além disso, assinalado esse local usando, eventualmente, cores ou materiais diferentes do resto do encapsulamento. O utilizador também espera que haja um botão para iniciar a leitura do DVD e que esse botão seja designado por "PLAY". É certo que se esse botão se chamar "GO", "START", ou algo semelhante, o utilizador acabará por perceber que é esse o botão a utilizar para essa função. Não é, no entanto, o que estava à espera. Estes exemplos podem parecer pouco importantes mas de facto têm peso na percepção que o utilizador tem da qualidade do produto, mesmo que isso aconteça de forma inconsciente.
- **Organização** – A interface deve estar organizada de acordo com as funções que o produto desempenha. Se a função de determinado produto for executar uma sequência de passos, como no caso de algumas máquinas usadas na indústria, os controlos e indicadores deve estar agrupados de acordo com o passo a que dizem respeito e dispostos sequencialmente da esquerda para a direita ou de cima para baixo no painel principal. Se a função executada

estiver organizada de forma hierárquica então os controlos e indicadores devem estar dispostos usando, por exemplo, uma árvore. Seja qual for o produto é importante que o utilizador não fique confundido com a função dos controlos que podem existir no painel e que em alguns casos podem ter que ser em grande número. Há certos casos, como por exemplo em teclados numéricos, onde o utilizador espera que os botões estejam dispostos de determinada forma. Neste caso é esperado que estejam dispostos segundo uma matriz com 3 colunas e 4 linhas e ordenados da esquerda para a direita e de cima para baixo (Figura 1.4). Organizações diferentes só servem para confundir o utilizador e diminuir a percepção de qualidade. Pode-se ainda hoje em dia encontrar, embora seja raro, produtos que o fazem como o exemplo mostrado na Figura 1.5.

Figura 1.4 – Fotografia de uma máquina de venda automática cujo teclado numérico apresenta uma disposição típica.

Figura 1.5 – Fotografia de uma máquina de venda automática cujo teclado numérico apresenta uma disposição atípica.

- **Indicações Gráficas** – O cérebro humano está bastante treinado para o reconhecimento de formas, tamanhos e cores. Isso deve ser usado na interface para transmitir indicações ao utilizador. O uso de um triângulo equilátero apontando para a direita é um símbolo reconhecido facilmente como significado "PLAY". Deve portanto ser usado em detrimento de outros símbolos para representar essa função. As cores também devem ser usadas para, por exemplo, distinguir diferentes grupos de controlos ou indicadores. O tamanho também é uma maneira de diferenciar os comandos de uma interface principalmente para dar a ideia da sua importância relativa. Naturalmente que o tamanho de um botão tem impacto na sua funcionalidade. Botões usados mais frequentemente devem ser maiores para facilitar o seu uso. Botões relacionados com funções menos importantes e/ou usadas menos vezes podem ser mais pequenos.
- **Retroacção** – É importante que o produto informe o utilizador do que está fazendo pois isso transmite ao utilizador uma maior sentido de controlo. Por outro lado informa o utilizador de que os seus comandos foram de facto recebidos ou de que está à espera de alguma indicação por parte do utilizador. No caso de funções que levam algum tempo a ser executadas é importante indicar o tempo previsto para a sua conclusão. Isso pode ser feito com o recurso, por exemplo, a uma barra de progresso ou um conjunto de leds. A ausência dessa indicação agrava no utilizador a sensação de falta de controlo levando às vezes mesmo ao desespero por se estar à espera de uma resposta há muito tempo e não se saber quanto tempo falta ainda para o fim.

- **Responsividade** – A responsividade traduz quão imediato é a retroacção obtida a um comando iniciado pelo utilizador. Uma resposta mais rápida transmite a ideia ao utilizador de que o produto é bom já que parece funcionar depressa. Mesmo que de facto as computações efectuadas não sejam rápidas, um tempo de resposta curto traduz-se por parte do utilizador numa impressão de facilidade de interacção. Veja-se o exemplo dos telemóveis modernos que são um misto de telefones e de assistentes digitais pessoais (PDA). Alguns utilizam o sistema operativo Windows Mobile da Microsoft. Esse sistema operativo, uma variante do sistema operativo vocacionado para os computadores pessoais, dá importância à funcionalidade, permitindo a instalação de inúmeras aplicações que podem correr em simultâneo. A interacção com o utilizador, no entanto, é lenta na medida que o tempo que leva ao atendimento das interrupções desencadeadas pelo carregar de uma tecla pelo utilizador é muitas vezes demasiado grande, porventura devido ao facto que o processador está ocupado a realizar outra função. Isso, que é um problema unicamente de software, acaba por transmitir a ideia que é o próprio hardware que não está a funcionar bem levando o utilizador a voltar a premir determinado botão por achar que o software não "recebeu" o comando. Os telemóveis da Apple (iPhone), por outro lado, utilizam um sistema operativo proprietário (iPhone OS) que além de só permitir a execução de uma aplicação de cada vez dá especial importância a que o utilizador receba a retroacção da sua acção o mais depressa possível. O funcionamento do iPhone é assim percebido como sendo mais ágil. Estudos mostram que se o tempo de resposta a uma acção iniciada pelo utilizador for inferior a 50 ms este percebe a resposta como sendo instantânea.
- **Design** – Um dos aspectos mais importantes na escolha de um dado produto por parte de um potencial comprador tem a ver com o seu design. Ninguém escolhe um produto por ter dentro de si um circuito electrónico especialmente bem concebido ou dimensionado. Escolhe com base na aparência, preço e em algumas especificações que considere mais importante. São no entanto esses circuitos electrónicos que fazem com que seja possível implementar as funções desejadas e que essas funções tenham um desempenho melhor do que as de produtos concorrentes.

1.2.7 Encapsulamento

Um dos aspectos mais visíveis é o seu aspecto visual exterior, em particular o seu encapsulamento. A sua escolha não deve ser feita *a posteriori* mas sim planeada desde o inicio tendo em conta a função do produto e o tipo de utilizadores alvo. O desenvolvimento do encapsulamento deve ser levado a cabo por equipas especializadas capazes de criar designs atractivos e funcionais mas que levem em conta, também, algumas condicionantes e compromissos em termos de engenharia e de produção em larga escala.

No que diz respeito ao design, à que ter em conta não só as tendências existentes na altura do desenvolvimento mas também o aspecto cultural. Diferentes culturas têm gostos em termos estéticos diferentes. Naturalmente que existe um compromisso entre custo de desenvolvimento e costumização de um mesmo produto para diferentes países. O design de um encapsulamento está relacionado não só com a forma e as cores utilizadas mas também com o posicionamento dos controlos e indicadores

e, por exemplo, o tipo de letra ou símbolos usados para as indicações. A Figura 1.6, por exemplo, mostra um aspirador robótico com um design moderno (2009).

Figura 1.6 – Fotografia de um aspirador robótico da Romba.

O desenho do encapsulamento está naturalmente condicionado pela função do produto e pelo ambiente onde vai ser usado. No caso de produtos de mercado global o custo é um dos factores mais importantes já que a prioridade é sempre criar um produto com um menor custo unitário possível de forma a aumentar a penetração no mercado. Noutras áreas, como na indústria, por exemplo, é dada bastante importância à robustez e durabilidade do encapsulamento. Na área médica a importância é dada à segurança. Na área militar as prioridades são a robustez e a fiabilidade.

As condições ambientais contribuem de forma decisiva para a escolha de um dado tipo de encapsulamento. Os equipamentos militares, por exemplo, têm que suportar não só temperatura e valores de humidade extremos mas também variações bruscas dessas condições, o que em geral coloca mais dificuldades devido à expansão e contracção dos materiais com a temperatura. Os equipamentos médicos têm que resistir a serem lavados e desinfectados regularmente. Os electrodomésticos têm que suportar quedas de alturas modestas (da ordem de 1 m). Em certas aplicações, como na indústria automóvel ou aeronáutica, mais importante que a resistência ao choque é a resistência à vibração.

O efeito do choque e vibração é mais acentuado na ligação entre cabos e conectores e entre componentes e placas de circuito impresso. Para evitar problemas deste tipo há que tentar minimizar o deslocamento de qualquer parte do encapsulamento e do seu conteúdo usando pontos de fixação suficientes em todos os componentes, especialmente os com maior massa, e usando amortecimento (uso de molas ou anilhas recortadas ou de borracha). Naturalmente os cabos e terminais de

componentes devem ser o mais curto possível. Uma das formas de evitar o uso de alguns cabos é soldar os controlos e indicadores directamente na placa de circuito impresso em vez de ligar um cabo com conector entre eles e a placa.

No caso dos conectores de ligação ao exterior existem diversas escolhas que têm que ser feitas. Conectores de aparafusar são mais resistentes à vibração do que os de enfiar ou encaixar (como os conectores BNC, por exemplo), mas são mais trabalhosos de usar. O uso de materiais especiais, como o ouro, pode ser uma solução quando a corrosão é uma preocupação como é o caso de produtos que são instalados no exterior.

O tamanho do encapsulamento está relacionado com a robustez e com a portabilidade. Encapsulamentos maiores são em geral mais robustos e portanto mais adequados em aplicações como as encontradas na indústria, por exemplo. Já a portabilidade requerida em produtos como telemóveis e computadores portáteis, por exemplo, determina que se tenham encapsulamentos com o menor tamanho possível sendo o limite determinado pelos componentes que é necessário colocar no seu interior. O uso de circuitos integrados, o uso de componentes de montagem em superfície e o uso de conectores miniatura são soluções adequadas nestes casos.

A robustez e portabilidade estão intimamente relacionadas, por outro lado, com o material utilizado. Materiais plásticos são mais leves mas menos resistentes enquanto o uso de metais, como o alumínio ou o aço inoxidável são escolhas que permitem uma maior robustez, embora à custa de um maior peso.

Os circuitos electrónicos usados em computadores/sistemas embebidos são em geral de baixa potência. Há, no entanto, que estudar a hipótese de serem necessários mecanismos de arrefecimento. Esses mecanismos passam pelo uso de dissipadores metálicos em certos componentes para dissipar o calor por condução, o uso de ventoinhas para circulação do ar e a correspondente dissipação de calor por convecção, ou mesmo o uso de sistemas de refrigeração. Estas duas últimas soluções, no entanto, acarretam um maior consumo energético.

As ventoinhas, por exemplo, são usadas em computadores de secretária e portáteis para dissipar o calor gerado principalmente pelo microprocessador. Nestas aplicações, especialmente nos computadores de secretária, a prioridade é dada ao desempenho. Já no caso dos telemóveis e assistentes digitais pessoais, são escolhidos processadores que dissipam menos energia (e também consomem menos energia) e que portanto dispensam o uso de ventoinhas. Note-se que em aplicações em que a robustez e resistência ao choque e vibração sejam extremamente importantes, o uso de ventoinhas, ou qualquer outro componente com peças móveis, é altamente desencorajado pois os rolamentos usados acabam por se deteriorar rapidamente.

1.2.8 Documentação

Qualquer produto desenvolvido necessita de ter documentação. Muitas vezes isso é feito depois de concluído o desenvolvimento do produto. Essa não é no entanto a forma mais correcta sendo mesmo praticamente impossível fazê-lo em projectos complexos.

Existem diferentes documentos que precisam ser criados e que variam no tipo de destinatário e na informação que contêm. Por um lado há documentos que são destinados a engenheiros envolvidos no desenvolvimento do produto, quer sejam elementos da mesma equipa e que trabalham em diferentes partes do projecto, quer sejam elementos de uma futura equipa cujo objectivo seja desenvolver uma nova versão do produto em causa ou mesmo criarem um produto inteiramente novo mas que beneficia do conhecimento adquirido pelos elementos da empresa em desenvolvimentos anteriores. Até no caso em que os elementos de uma equipa são substituídos, a documentação é essencial para que o novo elemento possa continuar o trabalho desenvolvido pelo colega que abandonou a equipa.

A experiência mostra que o tempo "perdido" na elaboração dessa documentação é recuperado mais tarde ao evitar perdas de tempo desnecessárias a tentar descobrir como algo foi feito ou a "reinventar" algo que já tinha sido criado na empresa.

Este tipo de documentação de engenharia pode-se classificar nos seguintes tipos:

- **Descrição do hardware** – Descreve o que é que determinado circuito faz, em termos funcionais, e como isso é implementado em termos do tipo de circuitos electrónicos usados.
- **Esquemas eléctricos** – Descrevem a conexão eléctrica dos diferentes componentes do circuito bem como as especificações eléctricas desses componentes (valores das resistências, etc.).
- **Memorandos** – São documentos trocados entre membros de uma empresa ou equipa e que servem para documentar o desenvolvimento de um certo produto através da explicação de onde vêm certos requisitos de projecto e porque é que certas soluções de engenharia foram aplicadas em detrimento de outras. É fundamental numa equipa grande em que o trabalho desenvolvido por cada um tem que ser integrado num todo por um outro membro da equipa e que portanto precisa saber as particularidades que cada módulo tem em termos de interface com os outros módulos.
- **Diários de Engenharia** – Descrevem os cálculos, simulações e ensaios efectuados e que foram usados para o desenho de certo produto. Isso pode dizer respeito, por exemplo, ao dimensionamento dos valores dos componentes eléctricos utilizados. Os diários de engenharia são também usados para descrever o resultado dos ensaios efectuados e como estes condicionaram a aceitação ou a reformulação do projecto de determinada solução de engenharia.
- **Folhas de Especificação** – Apresentam as especificações de determinado circuito com o intuito de informar o utilizador desse circuito de como deve conecta-lo a outros circuitos, como deve ser configurado, como deve ser ensaiado e em que condições deve operar.

Outra classe de documentação é a que se destina a ser fornecida ao utilizador de um produto. Uma má documentação pode prejudicar o sucesso de um produto que seja bom. Em geral não há maus

produtos que tenham uma boa documentação. Se o fabricante teve o cuidado em criar uma boa documentação, provavelmente também teve o cuidado de criar um bom produto.

Neste caso distinguem-se os seguintes tipos que podem, em alguns casos, ser integrados no mesmo documento:

- **Manual de Instalação** – Destina-se a explicar como determinado deve ser instalado de forma correcta.
- **Manual de Utilizador** – Descreve para o utilizador as diferentes funcionalidades do produto e como podem se acedidas e usadas.
- **Manual de Reparação** – Destina-se a técnicos responsáveis pela reparação do produto e incluem tipicamente a descrição dos ensaios a realizar para detecção da origem da falha bem como os esquemas eléctricos do produto de forma a permitirem a substituição dos componentes que eventualmente estejam danificados.

Há ainda a documentação com o intuito promocional cujo destinatário é um potencial comprador do produto. Nesta classe distinguem-se os seguintes tipos:

- **Folheto** – Serve para apresentar um dado produto chamando à atenção para as suas vantagens. Além de conter uma ou mais imagens do produto, contém geralmente a marca, modelo, aplicação e principais características que o distinguem de produtos concorrentes.
- **Apresentação Técnica** – Contém uma lista das especificações que permitem compará-lo com outros produtos com a mesma função criados pela mesma ou outra empresa.

2. HISTÓRIA E ORGANIZAÇÃO DOS COMPUTADORES

Este capítulo, na primeira de duas partes, versa sobre a evolução dos computadores desde o seu surgimento até hoje, com o intuito de dar a conhecer as ideias que foram surgindo para resolver os problemas encontrados na construção de computadores cada vez mais potentes. Outra dos objectivos desta primeira parte é o de introduzir a organização dos computadores modernos, tornando claro as suas componentes mais importantes. Os capítulos seguintes dedicam-se ao estudo e análise de cada uma dessas componentes, em particular à forma como as diferentes funções são realizadas em circuitos electrónicos.

O resto deste capítulo consiste numa revisão sobre as tecnologias usadas para construir processadores e sobre os circuitos electrónicos digitais que formam a base de um computador moderno.

2.1 Evolução dos Computadores

O primeiro passo para a compreensão de como são construídos os computadores electrónicos é o de perceber o princípio de funcionamento e a sua constituição. Essa constituição, ou arquitectura, tal como a encontramos hoje em dia, é um processo evolutivo muito dependente da tecnologia. Melhoramentos são feitos alterando e/ou complementando sistemas já existentes. Isso é feito através de novas ideias ao nível da arquitectura ou organização que surgem para resolver problemas ou deficiências existentes, ou pela utilização e aplicação de novas e mais desenvolvidas tecnologias.

Tendo em conta esse processo evolutivo, torna-se importante conhecer como surgiram as ideais e quais os avanços tecnológicos que conduziram à arquitectura dos computadores usada hoje em dia. Por um lado para aprendermos com essas ideais para uso próprio em futuros projectos e desenvolvimentos, nesta ou em outras áreas, mas também para sermos capazes de perceber o estado actual dos computadores e as suas principais limitações. É importante perceber as forças motrizes de índole moral, social e económica que levam a certos desenvolvimentos e a certas escolhas. Alguns exemplos são a criação de armas de destruição em massa, a procura do lucro através da redução do preço de produção e a criação de computadores para os países subdesenvolvidos.

2.1.1 Definição de Computador

A primeira pergunta que surge quando se inicia o estudo de computadores, inclusive da sua electrónica, é de onde vem a palavra "Computador". Essa palavra foi usada desde o início do século XVII até meados do século passado para designar a pessoa cujo trabalho era efectuar sucessões de cálculos/computações com o objectivo de determinar, por exemplo, tabelas de navegação, mapas de marés e posições de planetas para almanaques de astronomia. Note-se que um "Computador" não era uma pessoa que sabia efectuar cálculos mas sim uma pessoa cujo *trabalho* era efectuar uma sucessão de cálculos com um determinado fim. Só a partir do fim do século XIX é que começou a ser usada para designar a máquina que efectuava esses cálculos, sendo hoje em dia usada exclusivamente com esse significado.

Embora esta seja a origem da palavra "Computador", não se constitui uma definição do que é um computador. Na verdade não se encontra uma definição exacta e universalmente aceite do que é um computador. Por essa mesma razão é difícil identificar, à luz do que se entende hoje, qual foi o primeiro computador a ser concebido ou construído. A dificuldade está em distinguir o que é uma máquina de calcular e o que é um computador.

O ponto de vista que se adopta aqui baseia-se na origem da palavra "Computador". Um computador não é, no seu âmago, mais do que algo que serve para efectuar uma sequência de cálculos, tal como faziam os "Computadores Humanos". Naturalmente hoje em dia têm outras funções como representar, transmitir e armazenar informação. Note-se que essas funções, só por si, não qualificam um dispositivo como sendo um computador. Um tubo de raios catódicos não é um computador. Um DVD também não é um computador. Há autores que atribuem aos computadores a característica distintiva de serem programáveis. Há máquinas de calcular modernas em que possível "programar" qual a função matemática a usar para o cálculo. No entanto essa função é executada só uma vez aos dados introduzidos pelo utilizador. Para realizar uma sequência de cálculos o utilizador tem que inicia-los manualmente. Aqui, portanto, isso não é considerado como sendo um computador. A definição adoptada é

"Um computador é uma máquina capaz de efectuar de forma autónoma uma sequência de cálculos."

Os cálculos a efectuar e os dados iniciais podem ter que ser introduzidos manualmente, mesmo até através do ajuste de botões ou colocação de peças na máquina, como era feito antigamente. Mas depois de iniciado a computação esses cálculos são executados sem a intervenção do utilizador até estarem terminados.

A origem da palavra "Computador", por outro lado, elucida o porquê da razão de existirem computadores. Eles existem para fazer um trabalho que antes era feito pelo Homem, nomeadamente o cálculo. São bem sucedidos porque executam o cálculo mais rapidamente, não erram, não se cansam e podem facilmente ser replicados.

2.1.2 O Primeiro Computador

Desde há muito tempo que existem dispositivos que ajudam o Homem a fazer cálculos, como o ábaco (2400 AC), os ossos de Napier (1617) e régua de cálculo (1620). Máquinas que efectivamente são capazes de efectuar cálculos surgiram na Grécia antiga, por volta de 150 a 100 AC. O mecanismo de Antikythera e o Astrolábio, usados para estimar a posição dos planetas, do Sol e da Lua no céu são alguns exemplos. Esses dispositivos ainda não eram verdadeiramente computadores pois não eram capazes de fazer vários cálculos de forma autónoma. Faziam um cálculo único de cada vez, que tinha que ser iniciado manualmente pelo operador.

O "Relógio Castelo", inventado por Al-Jazari em 1206, e que era usado para indicar o zodíaco e as orbitas solar e lunar, podia ser reprogramado todos os dias de forma a ter em conta as diferentes durações do dia e da noite que se verificam ao longo do ano. Embora fosse programável, não era, no entanto, um computador.

Tendo em conta a definição apresentada, pode-se considerar que o primeiro computador a ser concebido foi o "Calculador Analítico." Esse computador, projectado por Charles Babbage em 1833 e cuja construção ele nunca chegou a terminar, era capaz de efectuar somas, subtracções, multiplicações e divisões e tinha a possibilidade de execução condicional e de realizar ciclos de processamento. Era um computador de aplicação geral, inteiramente mecânico, alimentado por um motor a vapor. Esse computador estava dividido em duas partes: uma onde eram efectuados os cálculos e outra onde se armazenavam os dados. Esta organização básica é idêntica aos computadores modernos. Eram usados três tipos de cartões perfurados: um para determinar a operação aritmética a realizar, outro para armazenar as constantes numéricas e outro para enviar e recuperar os dados da unidade aritmética. O resultado do cálculo era apresentado, também, através da perfuração de um cartão.

O uso de cartões perfurados foi inspirado pelos teares mecânicos desenvolvidos em 1801 por Joseph-Marie Jacquard (Figura 2.1). Esses cartões eram usados para determinar qual o padrão criado no tecido. A existência ou não de um furo no cartão determinava se determinado fio, de uma certa cor, seguia em frente ou não. Foi no entanto Babbage que teve a ideia que a presença ou não de um furo podia ser usada para representar uma grandeza abstracta.

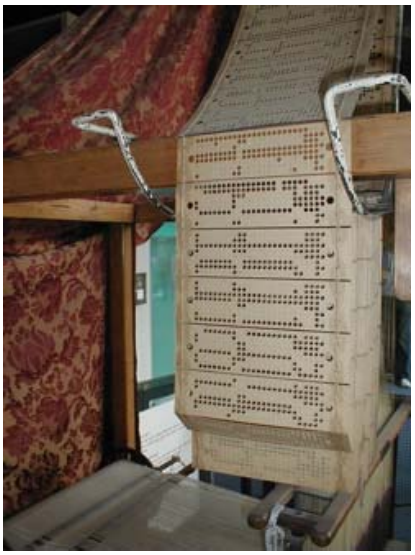


Figura 2.1 – Fotografia do tear de Jacquard e dos cartões perfurados usados para determinar o padrão criado no tecido.

Embora Babbage nunca tivesse chegado a construir o seu computador, o Museu da Ciência em Londres tem exposto uma parte desse computador construída em 1910, pelo seu filho mais novo (Figura 2.2).

Figura 2.2 – Fotografia de parte do Calculador Analítico, projectado por Charles Babbage em 1833. Esta é só uma parte desse computador construído em 1910 pelo seu filho mais novo. Tirada por Marcin Wichary. Distribuída através da licença Creative Commons Attribution.

O ritmo da evolução que hoje o computador conhece, tem crescido de uma forma contínua. Muitas vezes essas melhorias são incrementais, baseadas no avanço da tecnologia. Há no entanto, e de forma cada vez mais frequente, melhorias em termos da arquitectura e do funcionamento dos computadores e seus componentes.

O que hoje conhecemos como um computador e a forma como é projectado e implementado, que é o foco da disciplina de Electrónica de Computadores, é tão diferente da forma como o mesmo era feito nas primeiras máquinas a que podemos efectivamente chamar computadores que é quase incompreensível como dispositivos tão diversos podem pertencer à mesma categoria. Na verdade tal só é possível porque a definição de "Computador" é bastante lata.

Comparando um computador moderno com os diferentes computadores que foram sendo criados ao longo dos séculos quase tudo é diferente, desde as entidades físicas usadas (mecânicas vs. electrónicas), os componentes utilizados (relés, válvulas, transístores, circuitos integrados), o tipo de transístor utilizado (junção bipolar, MOSFET), a forma de representar a informação (analógica, digital em base 10, digital em base 2), o que é armazenado em memória (só programa, só dados, programa e dados), o tipo de utilizador (investigadores, empresas, particulares) e mesmo o âmbito das aplicações para que são construídos (muito específicas, de uso geral). Tudo isto tem mudado ao longo dos anos, naturalmente sempre em prol de uma maior rapidez de processamento e capacidade de cálculo e armazenamento.

2.1.3 De Mecânico a Electrónico

Por altura da invenção do 1º computador, o Calculador Analítico de Babbage, em 1833 ainda não havia sequer uma lâmpada que pudesse ser usada em larga escala. Isso só foi possível em 1879 com a invenção de Thomas Edison. É compreensível portanto que no pensamento de Babbage, quando planeava construir o seu computador, não estivesse o uso da electricidade. Nessa altura o que era

comum eram máquinas constituídas por engrenagem e alavancas impulsionadas por motores a vapor. O Calculador Analítico não é mais do que o uso da tecnologia existente para um fim completamente inovador.

Um século depois, no entanto, já a situação era bem diferente. Por essa altura o uso da electricidade já era universal. Por essa altura também, o Mundo estava envolvido numa guerra – a Segunda Guerra Mundial. As guerras em geral, mas essa em particular, são ambientes propícios a avanços tecnológicos em muitas áreas. Hoje em dia, por outro lado, é o capitalismo e a procura do lucro que impulsiona o avanço tecnológico.

Por altura da Segunda Guerra Mundial, um dos serviços mais importantes, para além da iluminação, era o telégrafo e o telefone. Por essa altura as centrais telefónicas já eram comuns. Um dos componentes fundamentais dessas centrais era o relé. Foi exactamente o uso comum de relés que levou o alemão Konrad Zuse, em 1941, a usá-lo para construir um computador, denominado de Z3, que é reconhecido como sendo o primeiro computador electromecânico. Dá-se aí então a primeira revolução na construção dos computadores que até aí eram mecânicos. O uso dessa nova tecnologia na construção de computadores permitiu abrir caminho para melhorias significativas, em particular em termos de rapidez. Qualquer dispositivo mecânico está irremediavelmente constrangido pela lei de inércia. Isso limita em termos práticos a capacidade de acelerar e indirectamente a rapidez de cálculo. O passo seguinte seria naturalmente o de abandonar por completo a mecânica, ou seja, fazer um computador completamente electrónico.

Com a guerra, Zuse estava isolado do resto do mundo, não estando a par dos desenvolvimentos que ocorriam, por exemplo, nos Estados Unidos da América. Aí John Vincent Atanasoff e Clifford E. Berry desenvolveram o *Atanasoff-Berry Computer* (ABC) que usava válvulas (mais de 300) e nascia assim o primeiro computador electrónico. Não era no entanto programável. O primeiro computador electrónico programável foi desenvolvido no Reino Unido por Tommy Flowers e denominado de Colossus. O seu propósito, como não podia deixar de ser, estava intimamente relacionado com a guerra – era usado para decifrar as mensagens cifradas dos Alemães.

Os computadores electrónicos persistem até aos dias de hoje, mas isto não significa que esta seja a tecnologia do futuro. No futuro os computadores poderão ser ópticos, ou até usar outras tecnologias ou princípios físicos que ainda não somos capazes de antever.

2.1.4 De Analógico a Digital

Ao mesmo tempo que os computadores passaram a ser electrónicos também a representação de informação sofre uma alteração marcante. Deixou ser feita de forma analógica para passar cada vez mais a ser feita de forma digital. Muito antes do surgimento dos primeiros computadores, a informação era representada de forma analógica, como no caso do astrolábio. Desde então muitas grandezas e medidas foram representadas e adoptadas como o comprimento, a pressão, a tensão eléctrica e a corrente eléctrica, suportadas em sólidos, como a pedra ou a madeira, no ar, na água e recentemente na carga eléctrica.

O MONIAC, por exemplo, é um computador analógico que usa a água (Figura 2.3). Foi criado em 1949 por Bill Phillips para modelar os processos económicos do Reino Unido [4]. É constituído por tanques de plástico transparentes conectados por tubos. Cada tanque representa um dado aspecto da economia e o fluxo de água entre os tanques representa o fluxo de dinheiro na economia. Na parte de cima do computador existe um tanque que representa o tesouro. A água flui desse tanque para outros tanques colocados mais abaixo representando, por exemplo, os sectores da saúde ou da educação. Para aumentar a verba gasta em saúde, por exemplo, basta abrir a torneira ligada ao tubo colocado entre o tanque do tesouro e o tanque da saúde. Por outro lado, a água dos tanques pode ser bombeada para o tanque superior (tesouro) o que corresponde à cobrança de impostos. Diferentes valores de velocidade de bombeamento correspondem a diferentes valores de taxa de imposto. Neste caso, por exemplo, a variável "taxa de imposto" é representada através da velocidade do fluxo de água na bomba, uma grandeza analógica. O fluxo de água entre os tanques, determinados pelas torneiras colocadas nos tubos de ligação é determinado por princípios económicos. O utilizador ajustava os diferentes parâmetros do computador para simular diferentes modelos económicos. Quando um dado modelo era viável o sistema atingia o equilíbrio e os seus parâmetros podiam ser lidos em escalas graduadas colocadas nos tanques.

Figura 2.3 – Fotografia do computador analógico MONIAC tirada no Banco Reserva da Nova Zelândia.

Este computador não só era usado para modelar a economia mas também como ferramenta de ensino para ilustrar os princípios económicos. Essa função era especialmente apropriada dado a forma visual como era apresentada a informação e o próprio cálculo.

O MONIAC pode ser considerado um computador, de acordo com a definição adoptada, pois é uma máquina que é capaz de efectuar uma sequência de cálculos de forma autónoma – a água flui constantemente de uns tanques para os outros e de volta ao tanque superior.

Os computadores analógicos exploram as semelhanças do comportamento de variáveis físicas como a posição e o movimento de rodas, ou o valor da tensão e corrente eléctricas em circuitos electrónicos, e o comportamento de outros fenómenos como a trajectória balística, a inércia, a ressonância, a transferência de energia, etc. Os computadores analógicos funcionam portanto por analogia com o sistema sobre o qual efectuam cálculos. De notar a origem comum das palavras "analógico" e "analogia" (assim como "análogo").

No contexto dos circuitos electrónicos, um sinal é dito analógico em oposição a um sinal digital. Essa distinção é feita na medida em que um sinal digital só assume um conjunto discreto de valores, numa dada gama, enquanto um sinal analógico pode assumir qualquer valor. Um sinal digital usado num computador pode assumir só dois valores a que estão associados dois símbolos, "0" e "1", nos quais se baseia a representação binária dos números.

2.1.5 Dos Relés às Válvulas

A utilização de relés em computadores, como o Z3, marca o início da era electrónica dos computadores. O passo seguinte, que determina a eliminação completa de componentes mecânicos para o cálculo foi a introdução das válvulas. Isso aconteceu com o computador Atanasoff-Berry (ABC) projectado em 1937 por John Vincent Atanasoff e Clifford Berry.

As válvulas não foram inventadas com o fim de desenvolver computadores. Como a maior parte dos avanços tecnológicos, elas surgiram de pequenas alterações a dispositivos existentes. No caso concreto das válvulas, pode-se viajar para trás no tempo, até 1857, para se encontrar o início do caminho que eventualmente iria permitir aplicações tão variadas como a lâmpada, o aparelho de televisão, o receptor de rádio, a radiografia e os computadores electrónicos.

Nesse ano, 1857, Heinrich Geissler, filho de um vidreiro alemão, inventou um tubo, mais tarde conhecido como tubo de Geissler, que consistia num cilindro de vidro com um eléctrodo em cada extremidade. O tubo era cheio de gás rarefeito (néon ou árgon), ar, liquido (mercúrio) ou mesmo sólidos ou minerais ionizáveis. A aplicação de uma alta tensão entre os eléctrodos fazia com que surgisse uma corrente eléctrica dentro do tubo. Essa corrente eléctrica, por sua vez, provoca a ionização do material dentro do tubo. Quando os electrões libertados voltam a recombinar-se com os iões formados, dá-se a libertação de fotões e uma consequente produção de luz. É o princípio usado hoje em dia nos reclames luminosos. A invenção deste tipo de tubo por Geissler deveu-se aos conhecimentos aprendidos com o seu pai e o seu interesse pela física.

A Figura 2.4 mostra o desenho de alguns tubos de Geissler construídos para efeitos decorativos. Esses tubos tinham formas variadas e complexas e eram eles próprios colocados dentro de um outro tubo de vidro[2].

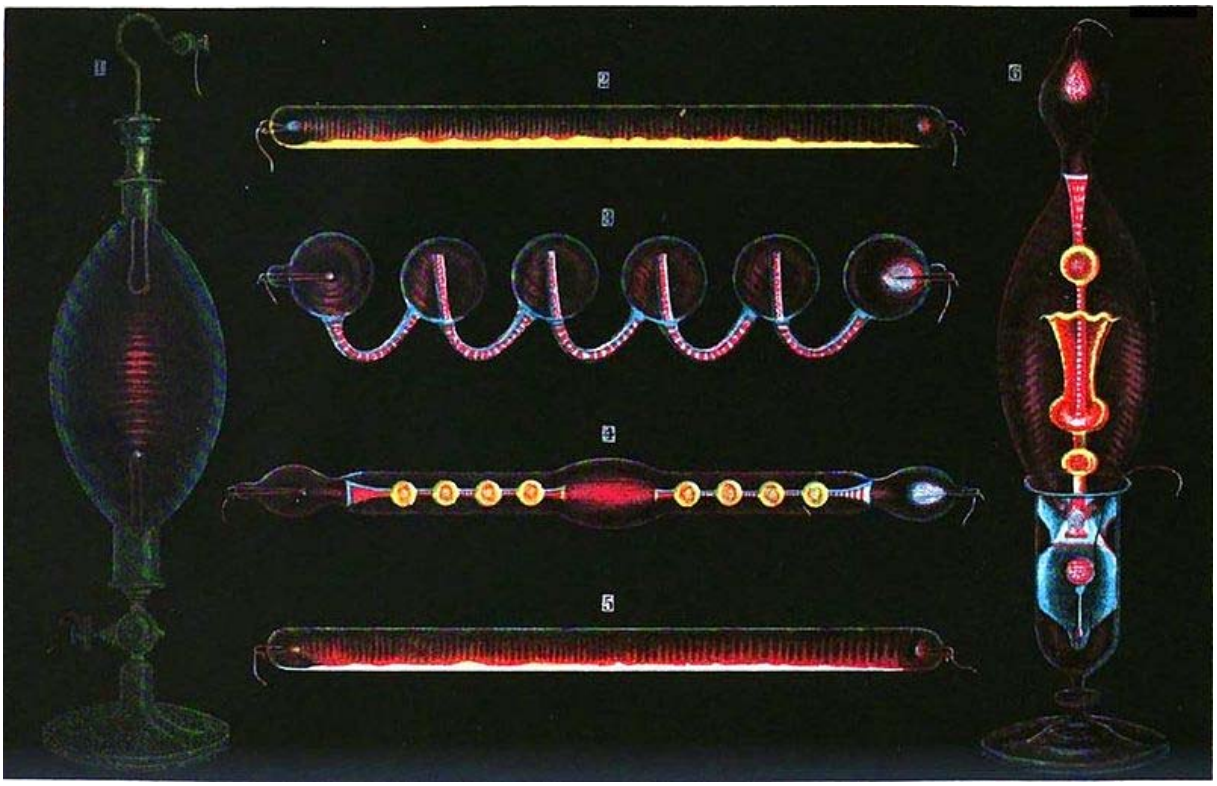


Figura 2.4 – Desenho de vários tubos de Geissler decorativos apresentado num livro de física de 1869.

Observou-se que, em certas condições, a extremidade positiva do vidro exterior também brilhava. Isso era tanto mais evidente quanto menos gás existisse dentro do tubo. A Figura 2.5 ilustra o fenómeno. À esquerda é representado um tubo com uma certa quantidade de gás. Verifica-se que luz é produzida pelo gás que se encontra entre o eléctrodo negativo (cátodo) e cada um dos dois eléctrodos positivos (ânodos). No caso da direita, em que o vácuo é mais forte (há menos ar), a parte do vidro oposta ao cátodo emite luz, sugerindo que existe algo que é emitido pelo cátodo e que embate no vidro. A esse "algo" foi dado o nome de raios catódicos, tendo mais tarde descoberto que se tratava de electrões.

[2] No YouTube pode-se observar um vídeo que mostra alguns tubos de Geissler em funcionamento. Endereço: *http://www.youtube.com/watch?v=D5nLA6PuzuQ*.

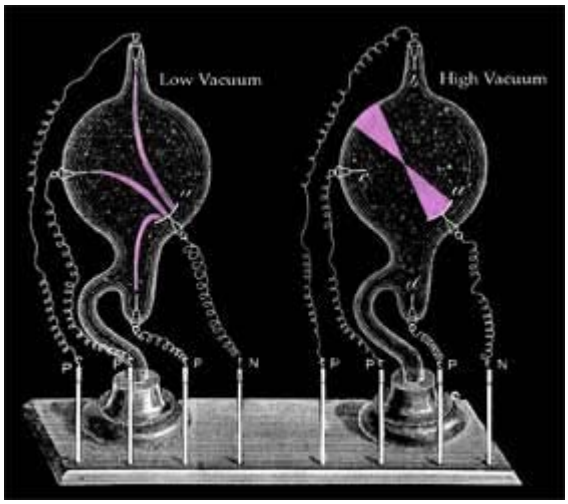


Figura 2.5 – Ilustração retirada do catálogo de James W. Queen & Co. (1891).

Para estudar este efeito, William Crookes construiu, por volta de 1869-1875, tubos especializados, conhecidos como tubos de Crookes, como o apresentado na Figura 2.6.

Figura 2.6 – Fotografia de um tubo de Crookes com uma Cruz de Malta no seu interior.

No tubo de Crookes a alta tensão aplicada entre os eléctrodos provoca a ionização do gás contido dentro do tubo. Os electrões libertados são acelerados pelo campo eléctrico existente dentro do tubo percorrendo uma trajectória rectilínea em direcção à extremidade oposta, onde embatem no vidro e provocam o surgimento de luz por fluorescência. No tubo da Figura 2.6 foi colocada uma Cruz de

Malta no caminho dos electrões de forma a demonstrar que de facto os raios catódicos viajavam em linha recta[3].

Quando a tensão aplicada entre os eléctrodos é muito alta (superior a 5 kV), os electrões adquirem uma energia cinética tão grande que quando embatem no vidro (ou no ânodo) produzem raios-X. Isso deve-se ao desvio brusco de trajectória que sofrem quando passam junto ao núcleo dos átomos, os quais têm uma carga positiva (*bremsstrahlung*[4]) ou então devido ao choque com os electrões das camadas interiores dos átomos. Esses electrões são então ejectados do átomo. Quando voltam a recombinar-se com o átomo libertam um fotão bastante energético (fluorescência de raios-X). Os princípios físicos estudados com os tubos de Crooke são os mesmos usados para criar os raios-X para as radiografias e os raios catódicos para os televisores tradicionais, comuns antes do surgimento dos televisores de ecrã plano.

Tanto nos tubos de Geissler como nos de Crookes, os electrões são criados por ionização do material que se encontra dentro do tubo. Existe no entanto outra forma de criar electrões livres chamada de *emissão termiónica*. Esse fenómeno foi inicialmente observado por Frederick Guthrie em 1873 numa esfera de ferro em brasa. Ele observou que se essa esfera estivesse carregada positivamente acabaria por perder essa carga com o tempo. Verificou também que se a esfera estivesse carregada negativamente isso não acontecia. O mesmo efeito foi estudado também mais tarde por Thomas Edison (1880) quando tentava descobrir a razão porque os filamentos que ele usava para criar uma lâmpada acabavam por se partir.

A emissão termiónica consiste na libertação de electrões de um material quando aquecido. O aumento de temperatura leva a um aumento da energia cinética dos átomos (e electrões) do material. Alguns electrões adquirem assim energia suficiente para se libertarem dos seus átomos.

Nas suas experiências, Edison introduziu dentro da lâmpada uma placa metálica a uma certa distância do filamento. Esse eléctrodo metálico foi ligado externamente ao filamento através de um galvanómetro tendo-se verificado que por ele passava uma corrente caso a placa metálica tivesse uma carga positiva em relação ao filamento, mas não passava qualquer corrente caso a carga fosse mais negativa (Figura 2.7). Além disso Edison verificou que a corrente através do galvanómetro aumentava rapidamente com o aumento da tensão aplicada entre a placa e o filamento.

[3] No YouTube pode-se observar um vídeo de um destes tubos de Crooke em funcionamento. Endereço: *http://www.youtube.com/watch?v=Xt7ZWEDZ_GI*.

[4] Bremsstrahlung, em alemão, significa "radiação que trava".

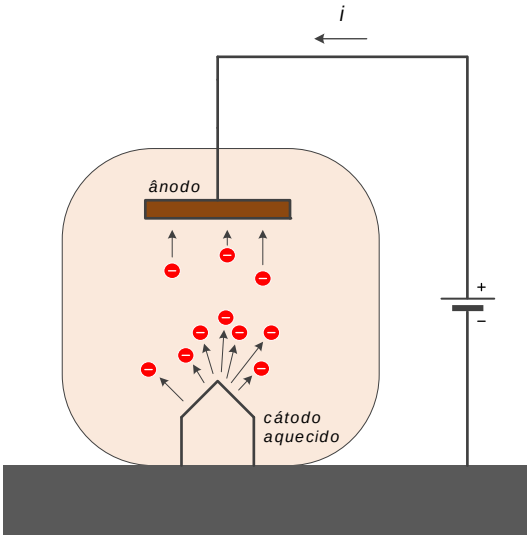


Figura 2.7 – Ilustração da constituição de um díodo usando um tubo de vácuo.

O facto de haver corrente num sentido e não no outro deve-se a que o filamento era aquecido e a placa não. Os electrões eram libertados no filamento e atraídos pela placa que se encontrava ligada ao terminal positivo da bateria ligado à placa. Quando o terminal positivo da bateria era ligado ao filamento, os electrões aí libertados eram repelidos pela placa e portanto nunca chegavam a saír da lâmpada (tubo de vácuo).

Este efeito (Efeito de Edison) foi utilizado por John Ambrose Fleming em 1904 para criar um dispositivo capaz de rectificar um sinal – o diodo. Fleming demonstrou que esse dispositivo podia ser usado para detectar ondas de rádio. O diodo, baseado num tubo de vácuo foi o primeiro dispositivo electrónico dando origem a toda uma era – a era da electrónica.

Em 1907 Lee De Forest colocou um fio condutor entre o filamento e a placa metálica e verificou que a tensão aplicada a esse fio condutor, mais tarde chamado de *grelha*, determinava a quantidade de electrões que chegavam à placa vindos do filamento. Era assim criado o *triodo* (Figura 2.8).

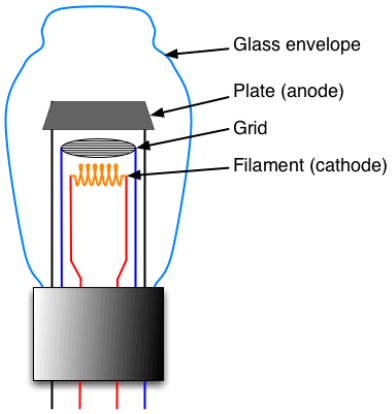


Figura 2.8 – Ilustração da constituição de tubos de vácuo (tríodo).

Esta foi a primeira vez que foi possível amplificar um sinal eléctrico utilizando um dispositivo não-mecânico. Muitos outros dispositivos se seguiram com o objectivo de controlar o fluxo de electrões. Note-se a distinção que se faz entre a electrónica e a electricidade. Esta última abrange a produção distribuição de energia eléctrica enquanto que a electrónica diz respeito ao controlo do fluxo de electrões com um determinado fim (amplificação, rectificação, filtragem, etc.). São exactamente este tipo de dispositivos, e a sua associação em circuitos complexos, criados com o objectivo de efectuar a execução autónoma de cálculos numéricos por uma máquina (o computador), que é o objectivo desta disicplina e o foco do resto deste documento – *a electrónica dos computadores*.

2.1.6 O Primeiro Computador Electrónico Digital

Em termos de resolução, um sinal analógico apresenta vantagens já que pode assumir uma infinidade de valores enquanto um sinal digital está limitado a um número finito de possibilidades em instantes de tempo definidos. Em termos práticos, no entanto, essa limitação é suplantada pela exactidão que se consegue na presença inevitável do ruído. Esse ruído, somado com um sinal analógico, causa invariavelmente um erro na informação por ele representada. Por outro lado, a presença de ruído no caso digital pode não alterar o símbolo associado e consequentemente a informação representada. Por exemplo, nos circuitos digitais do tipo TTL (*transistor-transistor logic*), o símbolo "0" encontra-se associado a qualquer valor de sinal entre 0 e 0,8 V e o símbolo "1" entre 2 V e 5 V. Por exemplo, um símbolo "0" representado com um sinal de 0,3 V, se afectado por ruído com um valor de 0,2 V, passará a ser um sinal com um valor diferente (0,5 V) mas representando o mesmo símbolo "0" (pois é menor do que 0.8 V). Percebe-se assim porque os sinais digitais são mais "resistentes" à presença de ruído.

A maior imunidade ao ruído apresentada pelos circuitos digitais suplanta, na maior parte das aplicações, as desvantagens de uma resolução finita. A resolução é determinada pelo número de bits usados para representar um número. É sempre possível melhorá-la, num sistema digital, aumentando o número de bits usados para representar os números. Hoje em dia, os processadores de uso geral, usados por exemplo em computadores de secretaria, são capazes de realizar operações aritméticas em números com 32 bits. O uso de 32 bits permite representar mais de 4 mil milhões de números diferentes. Isso é mais do que suficiente na maior parte das aplicações.

Pelas razões apresentadas, hoje me dia, e na maior parte dos casos, a informação é representada, armazenada e processada de forma digital.

Como foi referido, o computador Atanasoff-Berry (ABC), desenvolvido em 1937, é considerado o primeiro computador electrónico digital. Note-se que este computador não era programável nem era completamente electrónico. Só a unidade aritmética, ou seja, o "coração" do computador, era electrónica. O ABC, depois de completado, em 1941, continha 280 válvulas.

Este computador foi o primeiro a implementar quatro ideias fundamentais, ainda hoje usadas:

- Representação da informação de forma digital em base 2 (lógica binária);
- Uso de dispositivos electrónicos para a realização dos cálculos;

- Separação da parte onde eram armazenados os dados ("memória" em linguagem actual) da parte onde era efectuado o cálculo ("processador" em linguagem actual);
- Armazenamento da informação usando condensadores cuja carga precisava de ser regenerada periodicamente; Hoje em dia este tipo de memória é conhecido por DRAM (*Dynamic Random Access Memory*).

A memória do ABC era constituída por dois tambores que rodavam simultaneamente, uma vez por segundo, em torno no mesmo eixo. Cada tambor continha 1600 condensadores que estavam organizados em 32 bandas de 50 condensadores cada (Figura 2.9). Duas dessas bandas eram sobresselentes. Cada banda continha números de 50 bits.

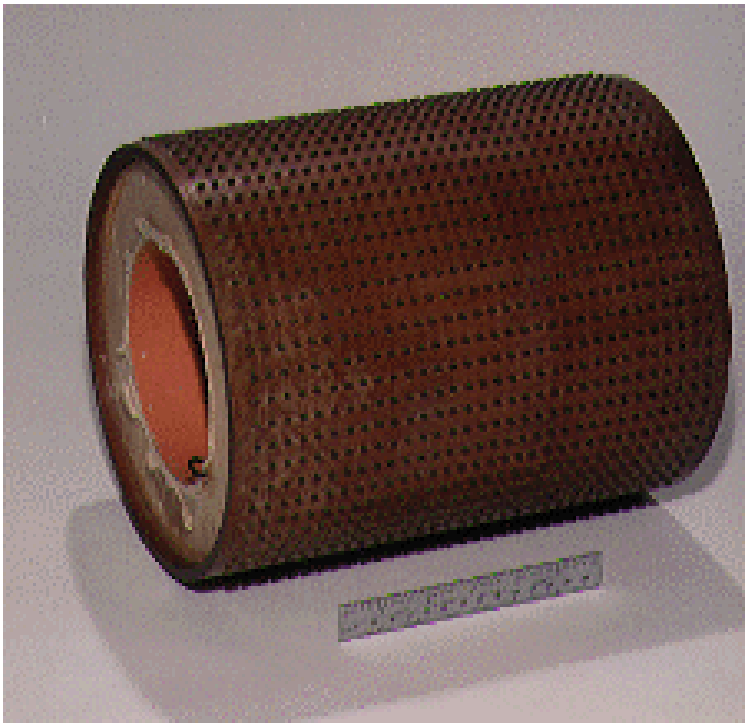


Figura 2.9 – Fotografia do tambor de memória do computador Atanasoff-Berry. É a única peça original ainda existente desse computador.

Os cálculos eram efectuados simultaneamente aos 30 números lidos dos tambores. Cada tambor tinha um dos argumentos da adição ou subtracção (Figura 2.10). Este funcionamento é hoje em dia conhecido como SIMD (*Single Instruction, Multiple Data*) e usado, por exemplo, em placas gráficas.

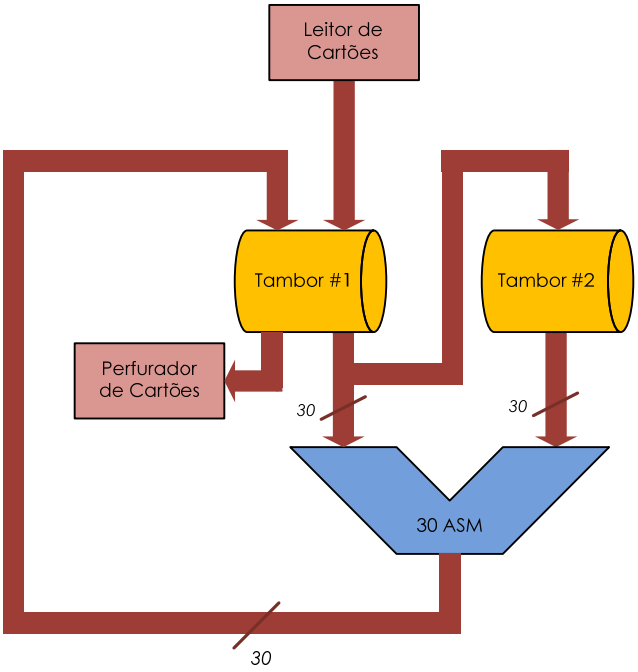
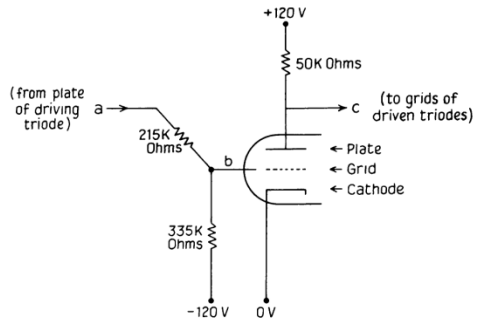


Figura 2.10 – Diagrama de blocos do computador Atanasoff-Berry ilustrando a ligação entre a memória (dois tambores) e a unidade de processamento que continha 30 módulos idênticos de soma/subtracção (ASM).

Atanasoff e Berry descobriram como realizar operações lógicas usando válvulas e a representação binária de números. No seu computador usaram um nível alto de tensão para representar um "0" e um nível baixo para representar um "1", o que hoje chamamos de lógica negativa. Os níveis de tensão de entrada e de saída das portas lógicas construídas não eram no entanto os compatíveis. Eram usados divisores resistivos para tornar esses níveis compatíveis. A Figura 2.11 mostra como era implementada uma porta NOT.



Input Voltage *a*	Grid Voltage *b*	Output Voltage *c*
+100	+15	+35
+35	−25	+100

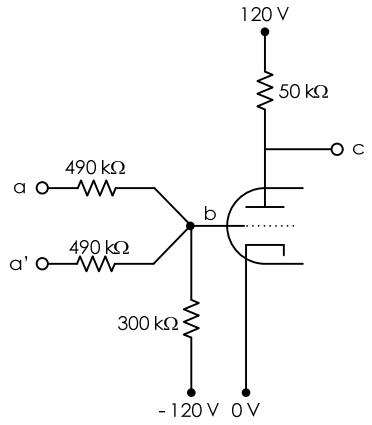
Input Bit *a*	Output Bit *c*
0	1
1	0

Figura 2.11 – Porta lógica NOT implementada com válvulas.

Ao sinal de entrada (ponto "a") é aplicado um divisor de tensão ligado a uma alimentação de -120 V. Quando a tensão de entrada tem um nível de tensão de 100 V ("0"), a saída do divisor de tensão fica com uma tensão positiva de 15 V. Essa tensão, aplicada à grelha da válvula, faz com que haja passagem de electrões do filamento para a placa e uma tensão de saída, no ponto "c", de 35 V, que é um nível baixo de tensão e corresponde a um "1". Um "0" à entrada dá portanto origem a um "1" à saída. No caso contrário, quando são aplicados 35 V à entrada ("1") a tensão à saída do divisor

resistivo (e entrada da grelha da válvula) passa a ser negativa (-25 V) o que faz com que não haja, praticamente, passagem de electrões do filamento para a placa. A tensão no ponto "c", devido à resistência de ***pull-up*** ligada aos 120 V, é nesse caso de 100 V o que representa um "0".

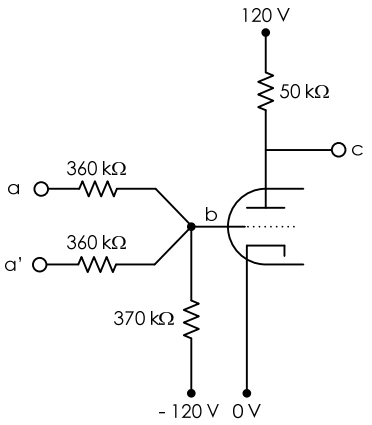
Da mesma forma eram construídas outras portas lógicas como portas NAND (Figura 2.12) e portas NOR (Figura 2.13).



Input Voltages		Grid Voltage	Output Voltage
a	*a'*	*b*	*c*
+110	+110	+5	+35
+110	+35	−15	+110
+35	+110	−15	+110
+35	+35	−35	+110

Input Bits		Output Bit
a	*a'*	*c*
0	0	1
0	1	0
1	0	0
1	1	0

Figura 2.12 – Porta lógica NOR implementada com válvulas.



Input Voltages		Grid Voltage	Output Voltage
a	*a'*	*b*	*c*
+110	+110	+35	+35
+110	+35	+10	+35
+35	+110	+10	+35
+35	+35	−15	+110

Input Bits		Output Bit
a	*a'*	*c*
0	0	1
0	1	1
1	0	1
1	1	0

Figura 2.13 – Porta lógica NAND implementada com válvulas.

Essas portas eram usadas para realizar um circuito capaz de somar e subtrair números. A Figura 2.14 mostra a tabela de verdade criada por Atanasoff e Berry para realizar a soma de dois números binários. Esse somador/subtractor efectuava a soma ou subtracção bit a bit, em série. Dados dois bits dos números a somar (α_i e β_i) e o transporte do bit anterior (γ_i), calculava o bit de soma (σ_i) e o bit de transporte para a soma seguinte (γ_{i-1}).

Variable Inputs			Control Settings		Intermediate Switching Points												Outputs	
Addend α_i	Augend β_i	Carry In γ_i	f	g	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_{10}	P_{11}	P_{12}	P_{13}	Sum σ_i	Carry Out γ_{i-1}
0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	1	1	1	0	0	1	0	1	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	0	1	0	1	1	0	1	0	0	1	0
0	1	1	0	1	1	0	0	0	1	1	0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	1	1	0	0	1	1	1	0	0	0	1	0
1	0	1	0	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1
1	1	0	0	1	0	0	1	1	1	0	1	0	1	1	0	1	0	1
1	1	1	0	1	0	0	0	1	1	1	0	1	1	0	0	1	1	1
Height	0				1			2			3	4		5			6	

Figura 2.14 – Tabela da verdade criada por Atanasoff e Berry para realização da soma de dois números binários.

A Figura 2.15 mostra o circuito lógico que implementa esta tabela de verdade (e a da subtracção).

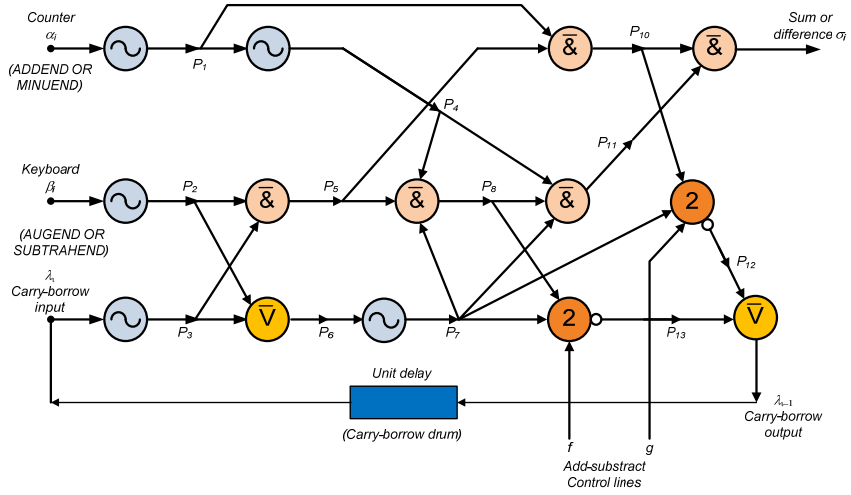


Figura 2.15 – Circuito lógico do somador/subtractor usado no computador Atanasoff-Berry.

Embora não fosse da mesma forma como se realizaria a operação hoje em dia, como se verá mais adiante, era um circuito que funcionava. Cada porta NOT, NAND e NOR, bem como uma outra função chamada de "2" e cuja saída tinha um nível alto se pelo menos duas entradas tivessem o nível alto, eram implementadas com válvulas e resistências dando origem ao circuito electrónico com 14 válvulas representado na Figura 2.16.

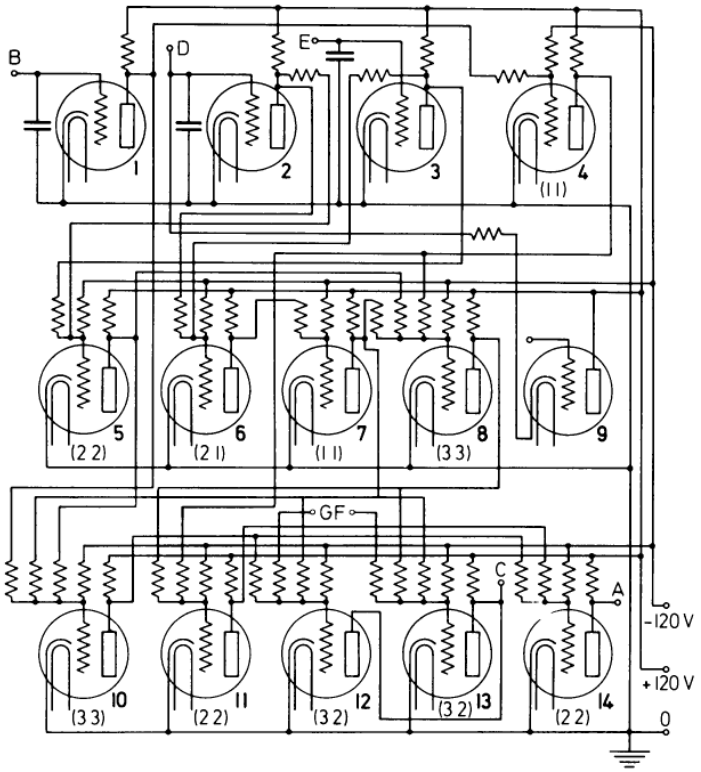


Figura 2.16 – Circuito eléctrico do somador/subtractor usado no computador Atanasoff-Berry.

Como se observa, a válvula 9 não era usada. Ela está representada no esquema eléctrico porque eram usadas válvulas duplas, isto, válvulas que tinham no mesmo tubo de vácuo, dois filamentos, duas grelhas e duas placas separadas. Eram assim usadas 7 válvulas duplas como se observa na Figura 2.17.

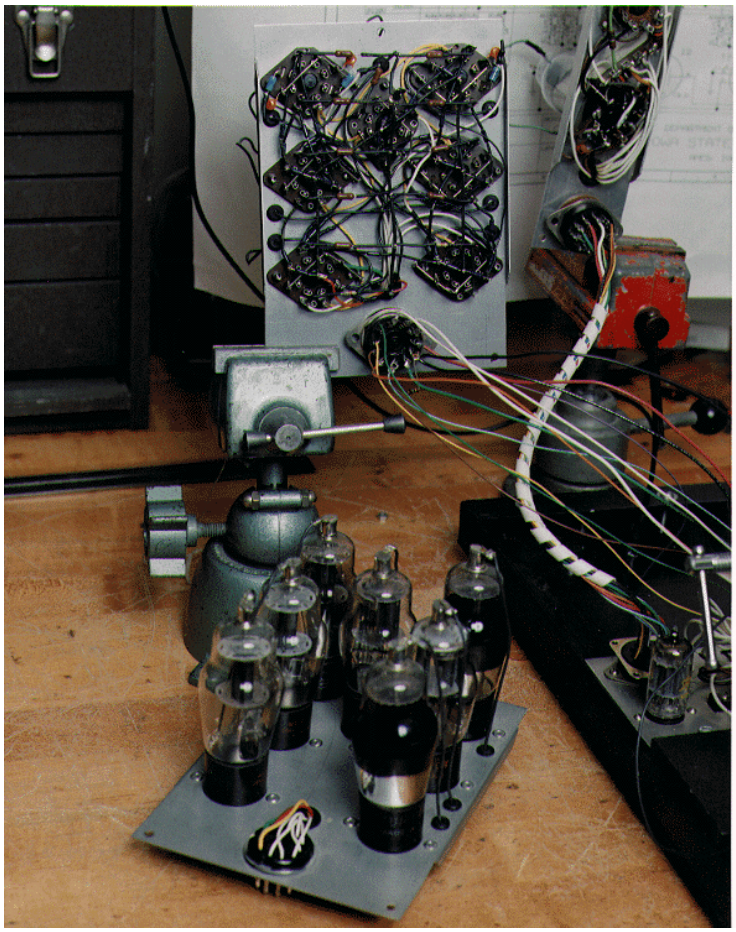


Figura 2.17 – Fotografia do circuito somador/subtractor do computador Atanasoff-Berry constituído por 7 válvulas duplas.

2.1.7 A Introdução do Transístor e do Circuito Integrado

O uso das válvulas melhorou um dos problemas dos computadores mecânicos e electromecânicos: a lentidão de operação. O ABC era capaz de efectuar 30 adições ou subtracções por segundo. As válvulas, no entanto, eram grandes, pouco fiáveis e consumiam muita energia. Alguns anos depois de completado o ABC, em 1947, uma nova tecnologia foi inventada que possibilitou a construção de computadores cada vez mais pequenos e mais rápidos – o transístor.

Em 1947, nos laboratórios Bell Labs, da AT&T, John Bardeen e Walter Brattain construíram um transístor feito de germânio, um material semicondutor, a que se dá o nome de *Point-contact transistor*. Esse transístor era constituído por um bloco de germânio, dopado com impurezas para formar um semicondutor do tipo-n, colocado sobre uma base condutora (terminal denominado de *Base*). Na face superior do bloco de germânio foram colocados dois contactos muito próximos um do outro e

feitos de ouro[5] (terminais denominados de "Emissor" e "Colector") como se ilustra na Figura 2.18. Ao fazer-se passar uma corrente eléctrica entre o emissor e a base criava-se uma pequena camada na superfície do bloco de germânio com uma deficiência de electrões. Verificou-se que a corrente que se fazia passar entre o emissor e a base afectava a corrente que fluía entre o colector e a base quando era aplicada uma tensão entre os dois. Verificou-se ainda que uma pequena variação no valor da corrente entre o emissor e a base levava a um grande variação no valor da corrente entre o colector e a base. Era assim possível usar este tipo de transístor para amplificar um sinal eléctrico.

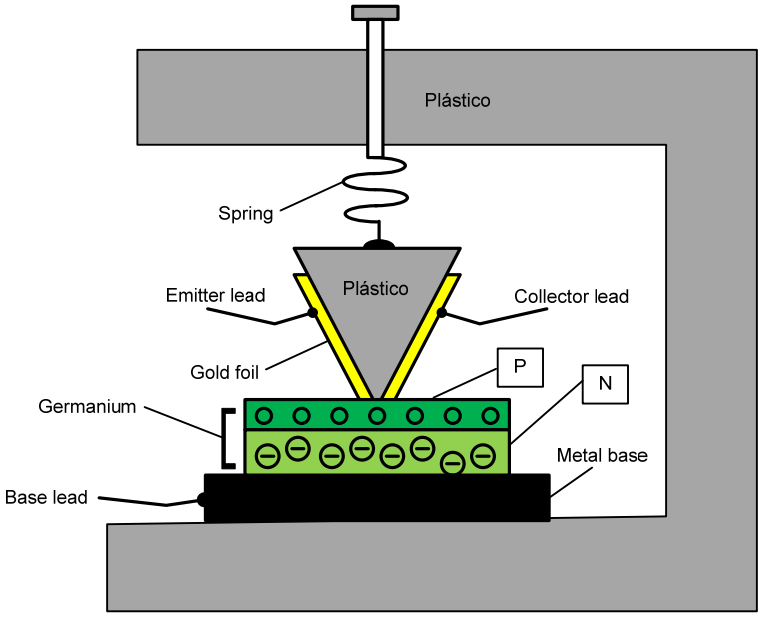


Figura 2.18 – Modelo de um transístor do tipo "*Point-contact*" construído por John Bardeen e Walter Houser Brattain em 1947.

A Figura 2.19 mostra uma réplica do primeiro transístor construído.

[5] Para construir esses contactos foi colocada uma folha de ouro sobre dois dos lados de um triangulo tendo-se depois raspado ligeiramente o vértice do triangulo para formar separar a folha de ouro em duas partes formando assim dois eléctrodos separados.

Figura 2.19 – Fotografia de uma réplica do primeiro transístor construído.

Este tipo de transístor foi usado por Dai Edwards e Doug Web, da Universidade de Manchester no Reino Unido, em 1953, para construir o protótipo do primeiro computador com transístores. Embora na altura fossem ainda menos fiáveis do que as válvulas, os transístores tinha a vantagem de consumirem menos energia (não havia nenhum filamento que necessitasse de ser aquecido) e a potencialidade de se tornarem mais pequenos e mais rápidos do que as válvulas.

Em 1948 William Shockley inventou o transístor de junção bipolar. Este transístor é criado pela junção de três blocos de material semicondutor dopados do tipo-n e do tipo-p. Os transístor do tipo NPN, como o representado na Figura 2.20, têm semicondutor do tipo-p no meio e semicondutor do tipo-n nas extremidades. Nos transístores PNP acontece o inverso.

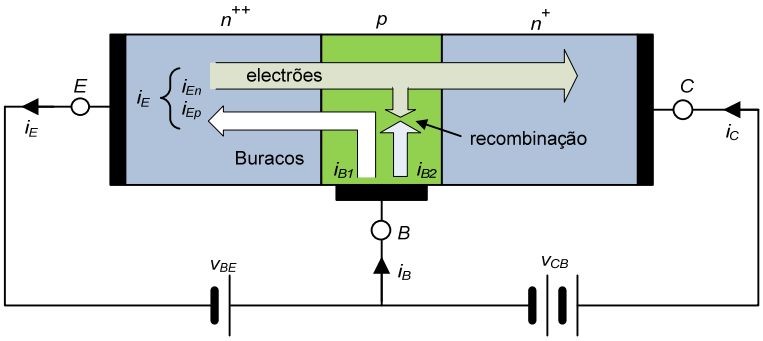


Figura 2.20 – Modelo do transístor de junção bipolar criado por William Shockley.

No caso dos transístores NPN, por exemplo, uma tensão positiva aplicada à junção base-emissor faz com que os electrões do emissor se desloquem em direcção à base por acção do campo eléctrico. A base é em geral muito fina de modo a que os electrões que aí chegam vindos do emissor consigam atingir o colector antes de se recombinarem com as lacunas que existem em maior número na base (semicondutor do tipo-p). Quando esses electrões chegam ao colector são de novo impulsionados pelo campo eléctrico criado pela tensão positiva aplicada à junção colector-base em direcção ao terminal colector do transístor. Assim sendo, e tendo em conta a Figura 2.20, os electrões viajam da esquerda para a direita, atravessando o transístor desde o emissor até ao colector. Quanto maior é a

corrente injectada na base do transístor maior é a corrente que atravessa o transístor do emissor ao colector. Essa pequena corrente da base é portanto capaz de controlar a quantidade de corrente entre o emissor e o colector de modo a que o transístor funciona como amplificador.

Os nomes dos terminais "emissor" e "colector" usados neste transístor advém do seu uso nas válvulas. O emissor é análogo ao filamento que emitia electrões quando aquecido devido ao efeito termiónico, e o colector é análogo à placa que recebia os electrões que atravessavam o vácuo dentro do tubo da válvula. Enquanto nesta o fluxo de electrões era controlado pela tensão da grelha, colocada entre o filamento e a placa, no transístor bipolar esse fluxo é controlado pela corrente injectada na base. A designação de "base" para este terceiro terminal do transístor bipolar vem da designação dada ao eléctrodo sobre o qual era colocado o bloco de germânio no transístor do tipo *point-contact*.

A ideia por detrás da válvula não é, portanto, muito diferente da ideia por de trás do transístor de junção bipolar. Os fenómenos físicos e os materiais envolvidos são diferentes. Isso no entanto é decisivo no uso do transístor de junção bipolar pois torna-o mais robusto, mais pequeno, mais rápido e mais fácil de fabricar em grandes quantidades. Na década de 60 e 70 a maior parte dos computadores usava este tipo de transístor ou uma variante.

A partir do fim da década de 50 vários tipos de circuitos lógicos usando transístores de junção bipolares foram propostos. Um primeiro tipo, chamado de RTL (*Resistor-Transistor Logic*), usava resistências e transístores. A Figura 2.21, por exemplo, mostra como é construída uma porta NOR usando lógica RTL. Basta uma das entradas (A ou B) estar com uma tensão alta para o transístor entrar em condução e levar a que a tensão no terminal de saída Q seja baixa. Se ambas as entradas estivarem no nível lógico baixo então o transístor fica ao corte e a tensão de saída fica no nível alto devido à resistência de *pull-up*, R_2. Esta montagem não é muito diferente da usada por Atanasoff e Berry no seu computador, como se observa na Figura 2.13 (nesse caso a lógica era negativa).

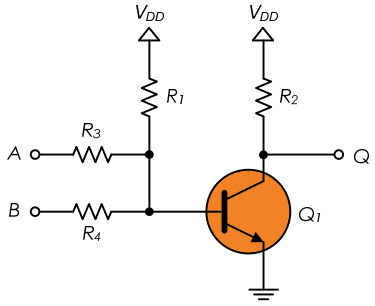


Figura 2.21 – Porta lógica NOR construída com lógica RTL.

Este tipo de lógica tinha a vantagem de necessitar de poucos transístores o que era fundamental nessa altura pois os transístores eram muito mais caros que as resistências. Por outro lado a grande desvantagem era a grande dissipação de energia devido à corrente necessária para polarizar convenientemente o transístor.

Rapidamente se desenvolveram outras lógicas com o objectivo de melhorar o desempenho dos circuitos lógicos. Tendo em conta que o preço dos díodos e transístores desceu rapidamente

aproximando-se do custo das resistências, criaram-se lógicas baseadas essencialmente nesses componentes como a DTL (*Diode-Transistor Logic*) e a TTL (*Transistor-transistor Logic*).

A Figura 2.22 mostra um exemplo de uma porta NAND usando lógica TTL. O andar de entrada é constituído por um transístor de junção bipolar com dois emissores. Quando uma das entradas (A ou B) estava no nível baixo o transístor de entrada conduzia, o que impunha um nível baixo de tensão na base do transístor de saída, que assim ficava ao corte, levando a saída Q a ter o nível alto de tensão.

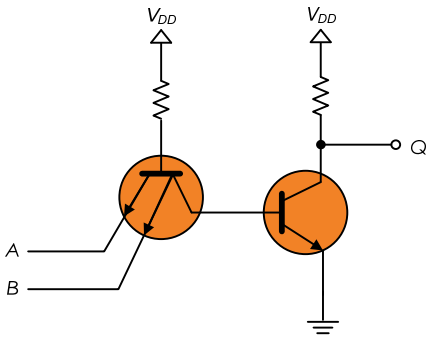


Figura 2.22 – Porta lógica NAND construída com lógica TTL.

A lógica TTL, introduzida em 1961, foi a mais utilizada a partir daí tendo-se criado variantes como *a low-power TTL(L)*, a *high-speed TTL(H)*, a *Schottky TTL(S)*, a *low-power Schottky TTL (LS)*, *a fast TTL (F)* e a *advanced-Schottky TTL (AS)*. A Figura 2.23, por exemplo, mostra um circuito integrado contendo 4 portas lógicas NAND usando lógica *low-power Schottky TTL*.

Figura 2.23 – Fotografia do circuito integrado 74LS00 contendo 4 portas NAND usando lógica *low-power Schottky TTL*.

O transístor bipolar viria a ser substituído para o desenvolvimento de computadores pelo MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*) proposto em 1925 por Julius Edgar Lilienfeld, mas cuja construção prática só aconteceu muito mais tarde. Estes transístores têm a vantagem de possuírem um muito baixo consumo, o que faz com que, hoje em dia, dominem quase na totalidade o mercado dos computadores.

Em 1958 deu-se um novo avanço na tecnologia dos computadores com a invenção do circuito integrado por Jack Kilby da Texas Instruments, o que lhe valeu o prémio Nobel em 2000. Passou a ser possível integrar, num único dispositivo, múltiplos componentes electrónicos. Sem isso não seria possível ter-se, como acontece hoje em dia, microprocessadores com várias centenas de milhões de

transístores num único dispositivo com poucos cm^2 de área. É justo referir que Robert Noyce também teve a ideia de construir um circuito integrado, meio ano depois de Kilby, mas que resolveu vários problemas que esse primeiro circuito integrado tinha. O integrado de Noyce era feito com silício (ao contrário do de Kilby que era de germânio) como são até hoje construídos os circuitos integrados.

Hoje em dia, portanto, a tecnologia amplamente utilizada para construir computadores baseia-se em transístores de efeito de campo (MOSFET) e circuitos integrados de silício, como se verá nas próximas secções.

2.2 O Computador Moderno

2.2.1 Transístor de Efeito de Campo

Os processadores, hoje em dia, são circuitos digitais construídos à base de transístores implementados em circuitos integrados. Os transístores mais usados hoje em dia são os do tipo MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*). Estes transístores são constituídos por 4 terminais: porta (G – *Gate*), substrato (B – *Body*), fonte (S – *Source*) e dreno (D – *Drain*)[6].

Quando aplicada uma tensão suficientemente elevada entre a porta e o substrato cria-se um canal entre a fonte e o dreno que possibilita a passagem de corrente (Figura 2.24). Quando não há tensão aplicada entre a porta e o substrato, não há fluxo de corrente entre fonte e dreno. Um MOSFET é assim capaz de funcionar como um interruptor controlado pela tensão aplicada entre a porta e o substrato.

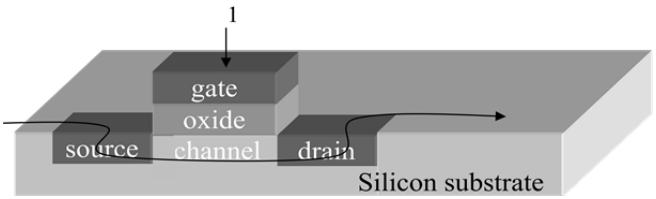


Figura 2.24 – Constituição de um MOSFET.

Os MOSFET são feitos de silício dopado. Existem dois tipos: nMOS e pMOS consoante o semicondutor usado nos terminais porta e dreno é do tipo-n ou do tipo-p respectivamente (Figura 2.25).

[6] Em transístores discretos a fonte e o substrato costumam estar ligados entre si. Nesses casos só saem para for do encapsulamento 3 terminais.

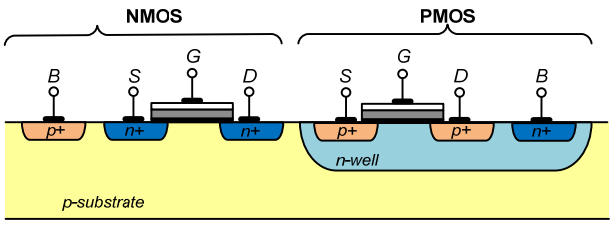


Figura 2.25 – Constituição dos transístores nMOS e pMOS.

Enquanto no caso dos nMOS o interruptor está fechado, ou seja, é permitida a passagem de corrente quando a tensão aplicada à porta é alta, no caso dos pMOS acontece o contrário. Quando é aplicada uma tensão à porta, os electrões são impedidos de fluir entre o dreno e a fonte. Os símbolos eléctricos destes dois tipos de MOSFET traduzem esta diferença com a presença de um círculo junto à porta no caso dos pMOS (Figura 2.26).

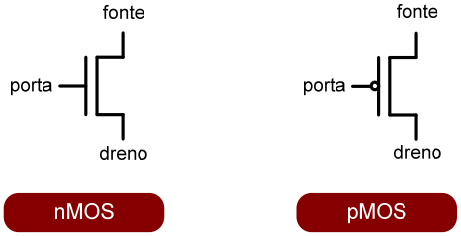


Figura 2.26 – Símbolos eléctricos dos transístores nMOS e pMOS.

Embora este tipo de transístores contenha no nome a referência ao uso de metal e óxido, referindo-se ao tipo de material usado no terminal porta e na camada dieléctrica por baixo dele respectivamente, hoje em dia é comum o uso de polisilício (silício cristalino) em vez de metal na porta. Recentemente, no entanto, alguns fabricantes de circuitos integrados, como a Intel, desenvolveram uma nova tecnologia de fabricação de MOSFET que usa de facto um metal no terminal da porta e substitui o dióxido de silício como dieléctrico por háfnio que é um material que tem uma maior constante dieléctrica (chamado pela Intel de "high-κ"). Isto permite evitar a passagem de electrões da porta para o canal, por efeito de túnel, que ocorre quando se tenta reduzir as dimensões dos transístores e em particular da espessura do dieléctrico abaixo dos 3 nm, o que corresponde a aproximadamente 12 átomos de espessura. Este tipo de transístor é usado na família de microprocessadores Core 2 e Core i7 da Intel criados com a tecnologia de 45 nm.

A principal vantagem do transístor MOSFET para uso em circuitos integrados é o seu baixo consumo energético. Na verdade só consome energia quando o canal é ligado ou desligado. Em regime estático não consome praticamente nenhuma energia.

2.2.2 Circuitos Digitais CMOS

Os MOSFET são usados para criar as portas lógicas que são a base dos circuitos digitais existentes nos computadores. Essas portas são criadas com uma tecnologia chamada de CMOS (*Complementary*

Metal Oxide Semiconductor). Esse nome traduz o facto de que são usados sempre pelo menos dois MOSFET complementares – um nMOS e um pMOS.

A porta lógica mais simples é a negação (NOT). No caso da construção de uma porta destas utilizando componentes discretos com base num MOSFET pode ser feita usando, por exemplo, o circuito Figura 2.27.

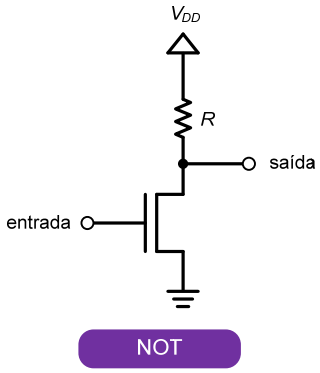


Figura 2.27 – Circuito eléctrico de uma porta lógica NOT usando componentes discretos.

Quando se coloca uma tensão nula na entrada o transístor não conduz, isto é, comporta-se como um circuito aberto. Devido à resistência, a tensão de saída será a tensão de alimentação (V_{DD}). Para isso acontecer é preciso que a corrente consumida pelo circuito a jusante seja pequena e que o valor da resistência seja alto. A implementação de uma resistência num circuito integrado requer, no entanto, uma área muito grande. Para resolver esse problema usa-se um outro MOSFET, neste caso do tipo pMOS, em vez da resistência como ilustra a Figura 2.28.

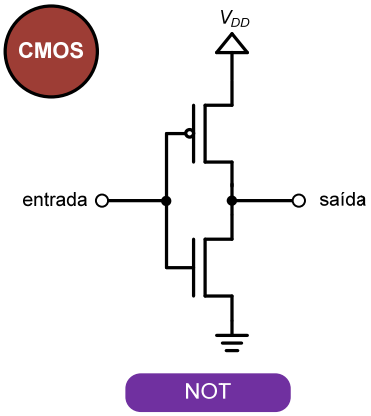


Figura 2.28 – Circuito eléctrico de uma porta lógica NOT usando tecnologia CMOS.

Quando a entrada está no nível lógico baixo o transístor nMOS está ao corte e o pMOS a conduzir e portanto o nível de tensão de saída é alto. Quando a entrada está no nível alto é o transístor nMOS que conduz o que faz com que a saída passe a estar no nível baixo.

A Figura 2.29 mostra como são construídas as portas NAND e NOR usando tecnologia CMOS. A plica é usada aqui para representar a negação embora alguns autores usem uma barra horizontal por cima das variáveis.

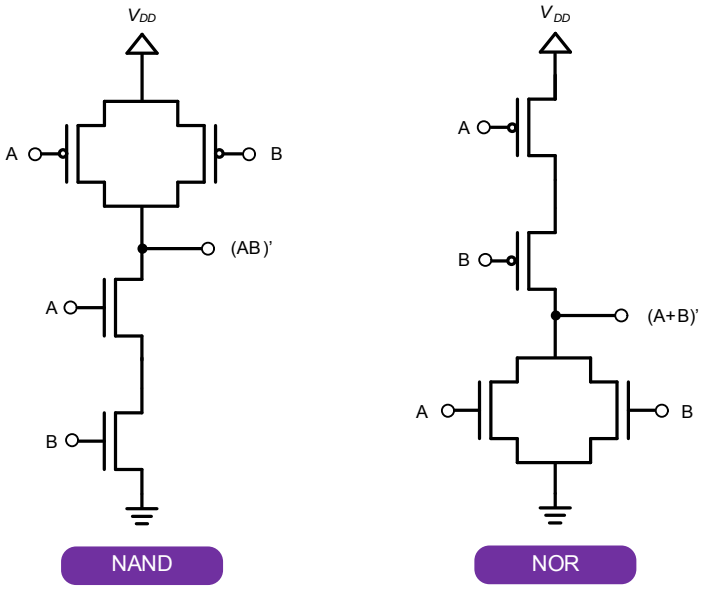


Figura 2.29 – Circuito eléctrico de uma porta lógica NOT usando tecnologia CMOS.

A Figura 2.30 mostra a disposição física de uma porta NAND realizada num circuito integrado.

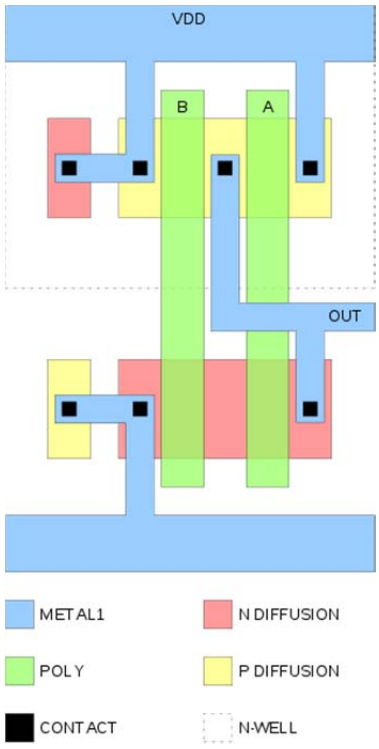


Figura 2.30 – Estrutura física de uma porta lógica NAND.

2.2.3 Lógica Combinatória

A Figura 2.31 mostra as diferentes portas lógicas, os seus símbolos, funções matemáticas e tabelas de verdade.

Figura 2.31 – Operações lógicas elementares.

Qualquer função lógica pode ser implementada usando estas portas lógicas. Como exemplo veja-se como é possível desenhar o circuito lógico dado pelo problema enunciado na Figura 2.32.

> *Y VALE 1 SE A VALER 1, OU B E C VALEREM 1.*
>
> *Z VALE 1 SE B OU C VALER 1, MAS NÃO AMBOS, OU SE VALEREM TODOS 1.*

Figura 2.32 – Enunciado de um problema exemplo onde se deseja obter o circuito lógico que implementa as funções descritas.

O primeiro passo consiste em escrever a tabela de verdade destas funções.

Tabela 2.1 – Tabela de verdade das funções descritas no problema exemplo.

Entradas			Saídas	
a	*b*	*c*	*y*	*z*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

As funções matemáticas podem obter-se da tabela de verdade somando, para cada linha em que a saída vale 1, um termo constituído pelo produto das variáveis de entrada, se tiverem o valor 1, ou a sua negação, se tiverem o valor 0. Por exemplo, a primeira linha em que *y* vale 1 é a 4 linha da tabela (abc = 011). O termo correspondente a esta linha é portanto $a'bc$. Fazendo o mesmo para o resto da tabela obtém-se

$$\begin{aligned} y &= a'bc + ab'c' + ab'c + abc' + abc \\ z &= a'b'c + a'bc + ab'c + abc' + abc \end{aligned} \quad (1)$$

Estas expressões matemáticas, embora correctas, não são as mais simples que é possível construir. Uma forma de as simplificar é usando mapas de Karnaugh. Esses mapas são, no fundo, outra forma de representar a tabela de verdade. É desenhado um rectângulo contendo do lado esquerdo uma ou duas das variáveis em causa e na parte de cima as variáveis que sobram. Esse rectângulo é dividido em linhas e colunas de acordo com o número de possíveis valores que as variáveis podem ter. No caso do problema apresentado existem 3 variáveis independentes (*a*, *b* e *c*). Assim sendo constrói-se um rectângulo com duas linhas (variável *a*) e 4 colunas (variáveis *b* e *c*) como se ilustra na Figura 2.33.

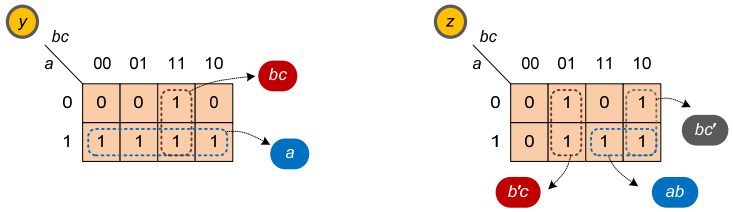


Figura 2.33 – Mapas de Karnaugh para o problema proposto.

A numeração das linhas e colunas é feita usando código Gray, isto é, só uma das variáveis pode mudar de valor entre colunas ou linhas adjacentes.

Cada área dentro do rectângulo é preenchida com 0 ou 1 de acordo com a tabela de verdade. De seguida desenham-se rectângulos (ou quadrados) que contenham só 1s, sejam o maior possível e que tenham uma largura e altura que sejam uma potência de 2 (2, 4, 8, …). Esses rectângulos podem sobrepor-se e podem sair de um dos lados para entrar no lado oposto (não é o caso apresentado na Figura 2.33). A cada um desses rectângulos vai corresponder um termo, chamado de mini-termo, na expressão final da função. Esses mini-termos são obtidos multiplicando as variáveis independentes cujo valor não varia ao longo da extensão do rectângulo (negada se esse valor que não varia for 0). Por exemplo, no caso da função *y*, a única variável que não varia ao longo do rectângulo azul é a variável *a*. Já no caso do rectângulo castanho nem *b* nem *c* variam. Esse mini-termo é portanto *bc*. O resultado que se obtém para o problema posto é portanto

$$\begin{aligned} y &= a + bc \\ z &= ab + b'c + bc' \end{aligned} \quad (1)$$

Como se observa estas expressões são muito mais simples do que as obtidas directamente da tabela de verdade (eq. (1)).

Finalmente o circuito lógico pode ser construído representando cada soma por uma porta OR e cada produto por uma porta AND. As negações das variáveis são naturalmente efectuadas com recurso a portas NOT (Figura 2.34).

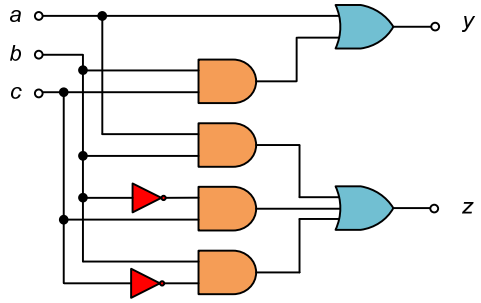


Figura 2.34 – Circuito lógico solução do problema proposto.

Note-se que para um número de entradas maior do que 6 não é possível sintetizar circuitos optimizados a partir das equações lógicas.

Existem várias ferramentas de desenho assistido por computador (CAD) para síntese de circuitos. A ferramenta Espresso, criada por Robert Brayton da IBM, por exemplo, é gratuita e fácil de usar[7].

Os circuitos como o representado na Figura 2.35 são circuitos de lógica combinatória pois a saída é obtida como uma combinação das entradas. Outros componentes que podem ser feitos da mesma forma são, por exemplo, multiplexers, descodificadores, somadores, comparadores, etc.

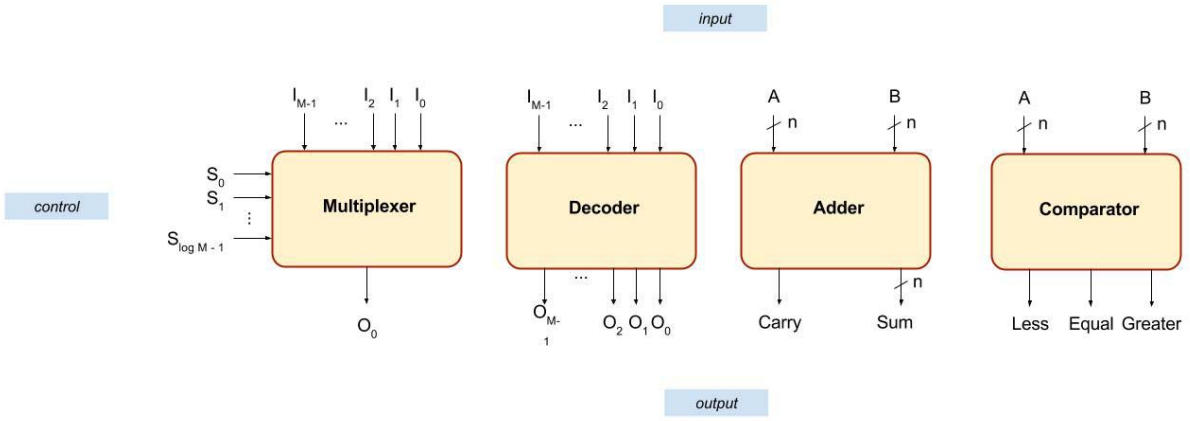


Figura 2.35 – Componentes de lógica combinatória.

[7] Está disponível para download por qualquer um na página da Internet da Universidade de Berkley em http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/

Um multiplexer selecciona qual das suas várias entradas é redireccionadas para a saída com base no valor de uma ou mais linhas de controlo. Funciona como um selector só que em vez de ser mecânico é electrónico (Figura 2.36).

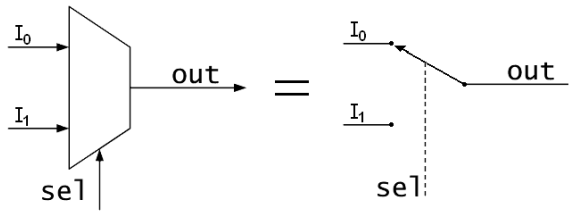


Figura 2.36 – Ilustração da equivalência entre um multiplexer com duas entradas e um interruptor.

Se existirem *m* entradas tem-se um multiplexer *m* x 1 e são necessárias $\log_2(m)$ linhas de controlo. Um multiplexer 16 x 1 tem 16 entradas, 1 saída e 4 linhas de controlo. A Figura 2.37 mostra o circuito lógico de um multiplexer 4 x 1.

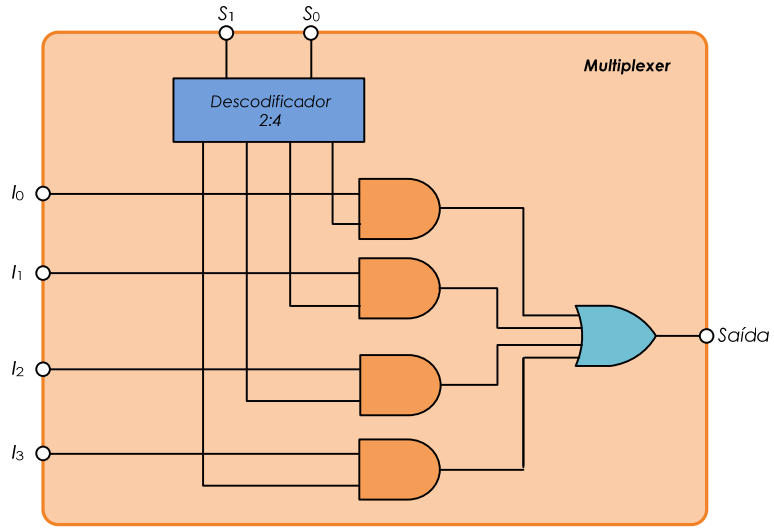


Figura 2.37 – Circuito lógico de um multiplexer 4 x 1.

Um descodificador possui diversas saídas. Uma, e só uma delas, é activada consoante o valor da palavra binária presente na sua entrada. Havendo *m* saídas têm de haver $\log_2(m)$ entradas. É comum um descodificador ter uma entrada de controlo extra, usualmente chamada "enable" que permite habilitar ou não o seu funcionamento. Enquanto estiver inactiva todas as saídas ficam no nível lógico baixo.

Um somador soma dois números binários com o mesmo número de bits (*N*). A saída é um número binário, também com *N* bits e um sinal indicando se ouve *overflow* (bit de transporte).

Um comparador compara o valor de duas palavras binárias com o mesmo número de bits activando uma de três linhas de saída conforme for o caso: *less*; *equal*; *greater*.

2.2.4 Lógica Sequencial

Há, no entanto circuitos digitais cuja saída depende das entradas actuais e das entradas em instantes anteriores. Esses circuitos, como registos e contadores, por exemplo, são chamados de circuitos de lógica sequencial (Figura 2.38).

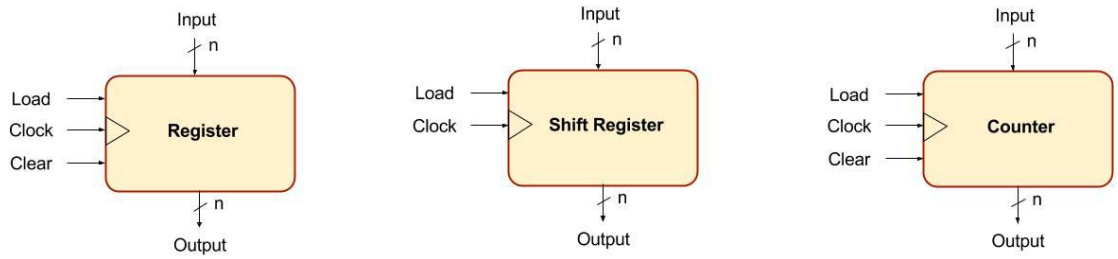


Figura 2.38 – Componentes de lógica sequencial.

Flip-Flop

Os circuitos de lógica sequencial são baseados em flip-flops que são no fundo circuitos capazes de armazenar 1 bit de informação. Existem vários tipos de flip-flops:

- Tipo SR – Armazenam um 1 quando a entrada S (*set*) está a 1 e armazenam um 0 quando a entrada R (*reset*) está a 1. São usados tipicamente como registos.
- Tipo T – O valor da saída troca de cada vez que o sinal de relógio tem um flanco caso a entrada T esteja no nível alto. São usados, por exemplo, para fazer contadores.
- Tipo JK – Comporta-se como um flip-flop SR. Quando ambas as entradas (R e S) são 1 o valor de saída troca de nível lógico.
- Tipo D – A saída assume o valor da entrada D em cada flanco ascendente ou descendente do relógio. Têm também uma entrada S para forçar a saída a 1 e uma entrada R para forças a saída a 0. São usados, por exemplo, para fazer registos de deslocamento.

A Figura 2.39 mostra o circuito lógico de um flip-flop do tipo SR. Se a entrada S estiver a e a entrada R estiver a 0 então a saída da porta NAND Y tem que ser 1 já que uma das suas entrada é 0 (a entrada R). Assim ambas as entradas da porta X estão a 1 o que faz com que a sua saída seja 0. Tem-se assim que a saída Q fica com o valor 1 e a saída Q' com o valor 0.

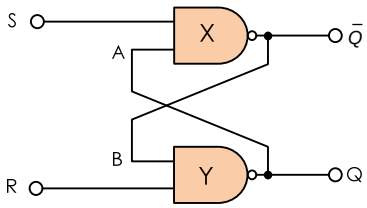


Figura 2.39 – Flip-flop do tipo SR.

O flip-flop do tipo D é baseado no flip-flop do tipo SR e em mais três portas lógicas (duas AND e uma NOT) que obrigam a que as entradas S e R do flip-flop SR sejam sempre diferentes (Figura 2.40).

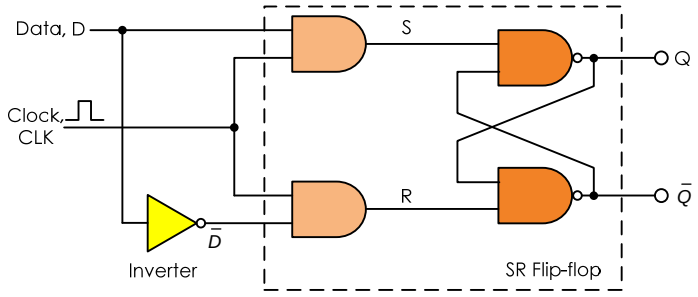


Figura 2.40 – Flip-flop do tipo D.

Registo

Um registo de *N* bits é feito usando, por exemplo, *N* flip-flops do tipo SR.

Registo de Deslocamento

Um registo de deslocamento desloca os bits de uma palavra digital um certo número de posições. Pode ser construído com flip-flops do tipo D encadeados. As entradas e saídas deste tipo de registo pode ser em série ou em paralelo. Isto é usado para converter um sinal de série para paralelo e vice-versa. A Figura 2.41 mostra um exemplo de um registo de deslocamento de 4 bits com entrada série e saída paralela.

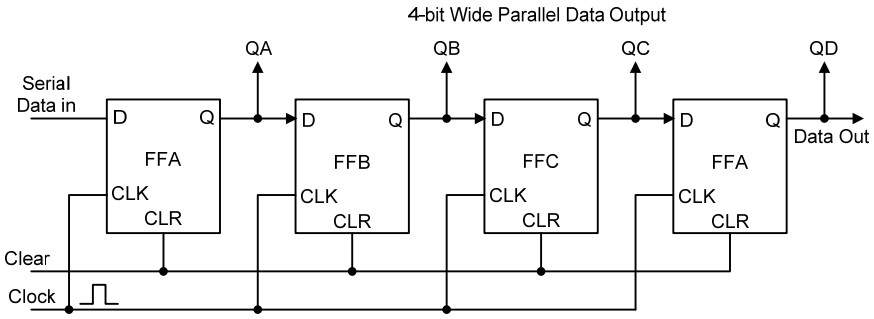


Figura 2.41 – Registo de Deslocamento de 4 bits.

A mesma configuração também funciona em modo de entrada série e saída série se for usada a saída Q do último flip-flop.

Máquina de Estados Finita

Um circuito digital importante em computadores é a Máquina de Estados Finita. Associa-se ao circuito lógico diferentes estados. No caso mais básico, o de um interruptor, o circuito pode ser descrito como tendo dois estados: aberto e fechado. A transição entre estados é condicionada pelo valor de variáveis externas (variáveis de entrada) e pelo estado actual da máquina. Essas transições são em geral implementadas usando lógica combinatória.

Considere-se, por exemplo, o problema da construção de um circuito digital para reduzir a frequência de um sinal de relógio 4 vezes.

> *A ENTRADA É UM SINAL DE RELÓGIO.*
>
> *A SAÍDA DEVE SER UM SINAL COM UMA FREQUÊNCIA 4 VEZES MENOR.*

Figura 2.42 – Enunciado de um problema relacionado com a máquina de estados finita.

Um circuito para resolver este problemas pode ser uma máquina de estados com 4 estados. A Figura 2.43 ilustra essa máquina de estados. Em três desses estados a variável de saída (*x*) é colocada a 0 e só num desses estado (estado 3) a variável de saída é colocada a 1. A máquina de estados transita de estado sequencialmente de cada vez que há um novo impulso no sinal de entrada (*a*).

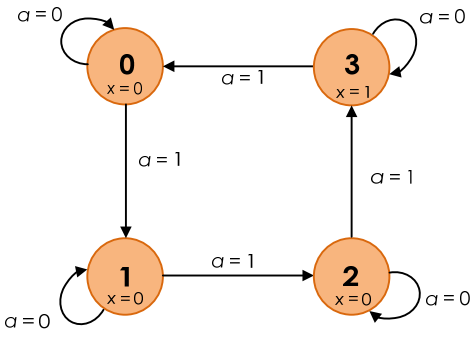


Figura 2.43 – Diagrama de estados correspondente ao problema apresentado.

O circuito lógico para implementar esta função necessita de um registo para guardar o valor do estado actual e um circuito de lógica combinatória para determinar as transições de estado (Figura 2.44). Como há 4 estados o registo tem de ser capaz de armazenar 2 bits. O circuito de lógica combinatória terá portanto 3 entradas (a entrada do sistema e os dois bits de estado) e 3 saídas (a saída do sistema e dois bit que determinam o próximo estado da máquina).

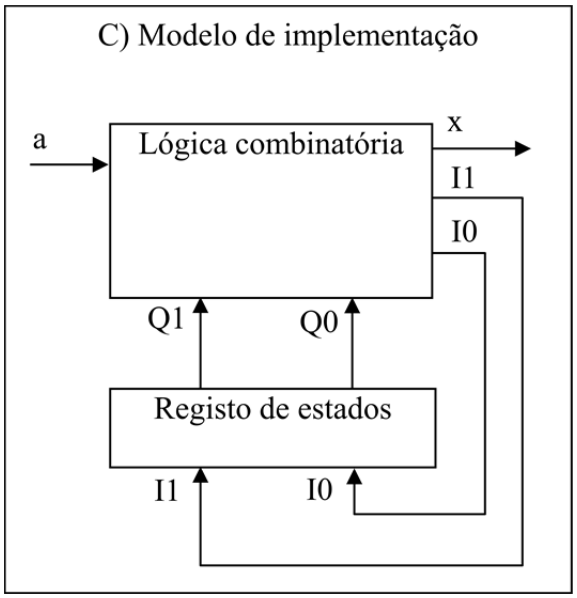


Figura 2.44 – Circuito lógico referente ao problema apresentado.

Observando o diagrama de estados (Figura 2.44) pode-se construir a tabela de estados (tabela de Moore) apresentada na Tabela 2.2.

Tabela 2.2 – Tabela de verdade das funções descritas no problema exemplo.

Entradas			Saídas		
Q1	*Q0*	*a*	*I1*	*I2*	*x*
0	0	0	0	0	0
0	0	1	0	1	
0	1	0	0	1	0
0	1	1	1	0	
1	0	0	1	0	0
1	0	1	1	1	
1	1	0	1	1	1
1	1	1	0	0	

Usando mapas de Karnaugh é possível determinar as funções lógicas a realizar pela lógica combinatória (Figura 2.45).

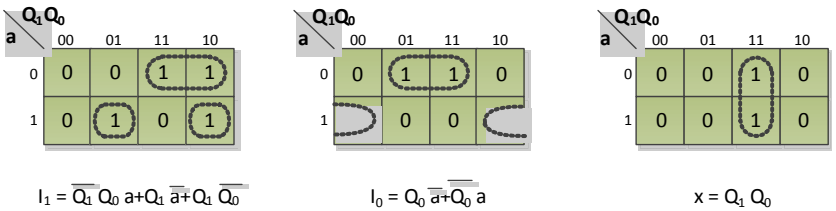


Figura 2.45 – Mapas de Karnaugh usados para obter as expressões matemáticas do problema apresentado.

Obtêm-se assim as seguintes expressões matemáticas:

$$\begin{aligned} l_0 &= Q_0\overline{a} + \overline{Q_0}a \\ l_1 &= \overline{Q_1}Q_0 + Q_1\overline{a} + Q_1\overline{Q_0} \\ x &= Q_1Q_0 \end{aligned} \quad . \qquad (1)$$

Essas funções podem ser implementadas com o circuito da Figura 2.46.

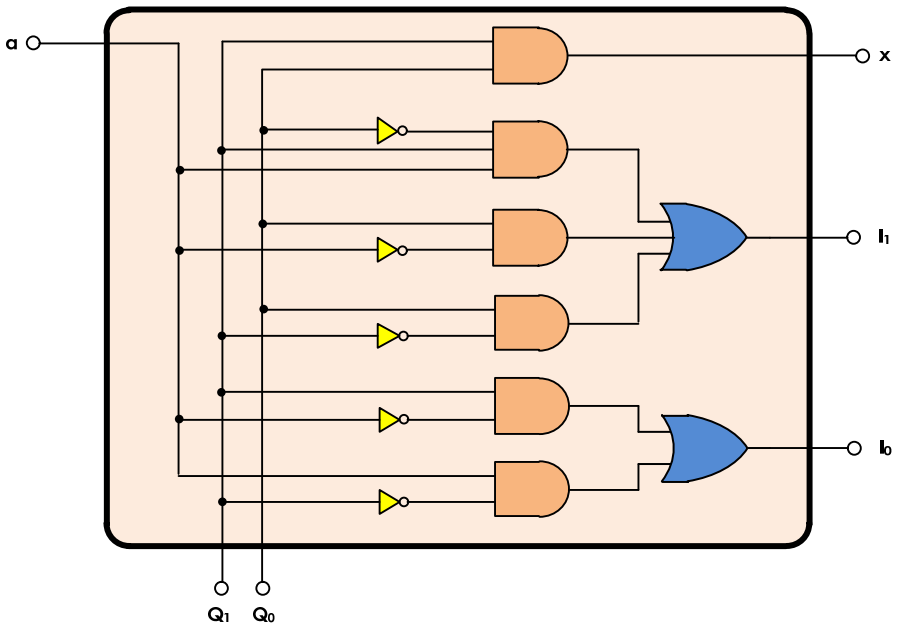


Figura 2.46 – Circuito de lógica combinatória a usar na solução do problema apresentado.

2.2.5 Organização Interna

Finalmente, nesta breve resenha dos avanços tecnológicos efectuados desde o surgimento do primeiro computador, há que referir como evoluiu a organização interna dos computadores e que motivos levaram a que os computadores sejam construídos da forma como o são actualmente.

O primeiro computador, o Calculador Analítico, projectado por Charles Babbage em 1833, era programável através do uso de um conjunto de cartões. Existiam três tipos de cartões: com números; com a indicação da operação aritmética a realizar; e com outras instruções como o carregamento de dados de e para a memória. A programação consistia na escolha dos cartões usados e na sua ordem.

O primeiro computador electrónico, o Atanasoff-Berry (1942) não era programável. Foi construído com o objectivo de resolver sistemas de equações não sendo capaz de ser usado para nenhuma outra função.

Outros computadores mais recentes, nomeadamente a maioria dos computadores criados no início da década de 1940, podiam ser usados para mais do que um objectivo mas necessitavam para isso de ser reconfigurados manualmente através da conexão de cabos e interruptores. Alguns exemplos são o Colossus Mark 1 e 2 (1944), usados pelos Ingleses para quebrar os códigos usados pelos Alemães para codificar as suas mensagens durante a Segunda Guerra Mundial, e o ENIAC (1946) criado para ser usado pelo exército dos EUA para calcular tabelas de artilharia, mas que foi usado primeiro para efectuar cálculos relacionados com o desenvolvimento da bomba de hidrogénio.

A configuração manual do computador era de facto uma operação bastante demorada e um ponto fraco desses computadores. Em 1944 a IBM desenvolveu o ASCC (*Automatic Sequence Controlled Calculator*), também conhecido por Mark I na Universidade de Harvard para onde foi enviado quando concluído. O ASCC era um computador electromecânico em que o programa era lido de uma fita de papel perfurada (Figura 2.47).

Figura 2.47 – Fotografia de uma fita de papel perfurada.

O uso de fitas de papel perfuradas permitia a execução fácil de diferentes programas dispensando a reconfiguração física do computador. As fitas usadas tinham 24 colunas que estavam organizadas em 3 conjuntos de 8 colunas cada. Um desses conjuntos continha o número indicativo da operação a ser executada (cada unidade aritmética tinha um número identificativo). O segundo conjunto tinha o número do sítio de onde era lido o dado para realizar a operação e que podia ser o acumulador, um registo de entrada ou um conjunto de interruptores onde eram introduzidas manualmente as constantes. O terceiro conjunto indicava onde devia ser colocado o resultado do cálculo efectuado.

À medida que os computadores vão evoluindo, tal como acontece em qualquer sistema complexo, há a necessidade de encará-lo como sendo constituído por várias subunidades mais pequenas para facilitar o projecto e tornar a implementação e o seu ensaio mais simples. Cada subunidade por sua vez pode ser constituída por outras subunidades ainda mais pequenas. Essa organização hierárquica encontra paralelo no universo que nos rodeia e na própria maneira de funcionar do nosso cérebro. As árvores são constituídas por raízes, tronco e ramos. Os ramos, por sua vez, têm folhas e frutos. Os frutos têm casca, polpa e semente. Da mesma forma, o funcionamento do cérebro, nas suas diferentes vertentes, está organizado de forma hierárquica. O reconhecimento da cara de uma pessoa, por

exemplo, é baseado no reconhecimento do cabelo, olhos, nariz e boca e da sua relação especial. O reconhecimento da boca, por sua vez, é baseada no reconhecimento das formas que a compõem, formas essas que são identificadas a partir da detecção pelo córtex visual de linhas e curvas.

No caso do computador também podemos descrever a sua organização na forma hierárquica. O computador, o primeiro nível da hierarquia, é constituído por diferentes módulos (subunidades) com funções distintas. Tendo em conta a principal função de um computador, a realização de cálculos, facilmente se chega à necessidade de ter módulos funcionais encarregues da realização dos cálculos propriamente ditos. Para fazer cálculos é necessário fornecer os operandos, geralmente designados por dados. É por isso necessário um módulo de entrada que permita ao utilizador fornecer os dados ao computador. O resultado do cálculo em si deve, por sua vez, ser apresentado ao utilizador. É portanto necessário também um módulo de saída. O que distingue um computador de uma máquina de calcular é a capacidade de efectuar, de forma automática, uma sequência de cálculos e não um só cálculo de cada vez. Isso leva à necessidade de armazenamento temporário dos dados entre cada cálculo, o que corresponde a um módulo de memória (Figura 2.48).

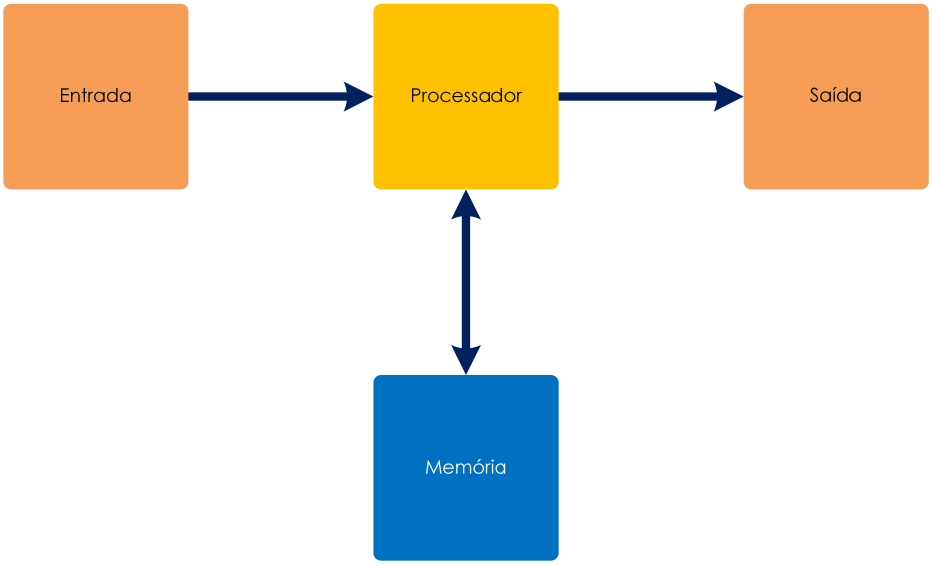


Figura 2.48 – Módulos de um computador.

Estes termos são os utilizados hoje em dia para designar as diferentes partes dos computadores. No início da era dos computadores outros termos eram usados, dependendo do computador em causa. Outros termos usados hoje em dia são "instrução", que significa o tipo de cálculo ou operação a realizar pelo computador (soma, leitura de uma certa posição de memória, saltar para uma certa instrução, etc.), "programa", que se refere ao conjunto das instruções que devem ser executadas, e "dados", designados por operandos e os resultados das operações.

No caso do Atanasoff-Berry os cálculos efectuados pelo processador eram sempre os mesmos. No caso do Colossus Mark 1 e 2 e do ENIAC os cálculos efectuados podiam ser alterados mas tal era feito pela reconfiguração física do próprio processador. No Harvard Mark 1 os cálculos a efectuar eram

fornecidos pelo utilizador através de uma fita perfurada colocada no dispositivo de entrada do computador.

Em 1944 John Macuchly e Presper Eckert propuseram a construção de um computador em que tanto as instruções como os dados fossem armazenados no mesmo dispositivo de memória (*stored-program computer*) – o EDVAC. John von Neumann, que se tinha juntado à equipa que construía o EDVAC, publicou em 1945 um documento onde descrevia a arquitectura desse computador e a ideia de armazenar o programa e os dados num mesmo dispositivo de memória[8].

Entretanto Frederic C. Williams desenvolveu um dispositivo de memória capaz de armazenar 2048 bits (Tubo de Williams). Williams, em conjunto com Tom Kilburn e Geoff Tootill construiu um computador para demonstrar esse dispositivo de memória a funcionar. Esse computador, o SSEM (Manchester Small-Scale Experimental Machine), completado em 1948 armazenava o programa e os dados na mesma memória de leitura/escrita de acesso aleatório. Mais tarde, em 1949, outros computadores, como o EDSAC, o Manchester Mark 1 e o CSIRAC, também usaram essa filosofia que passou a ser praticamente a norma. A esta arquitectura dá-se o nome de *Arquitectura de von Neumann*, em grande parte devido ao documento publicado em 1945 sobre o EDVAC (Figura 2.49).

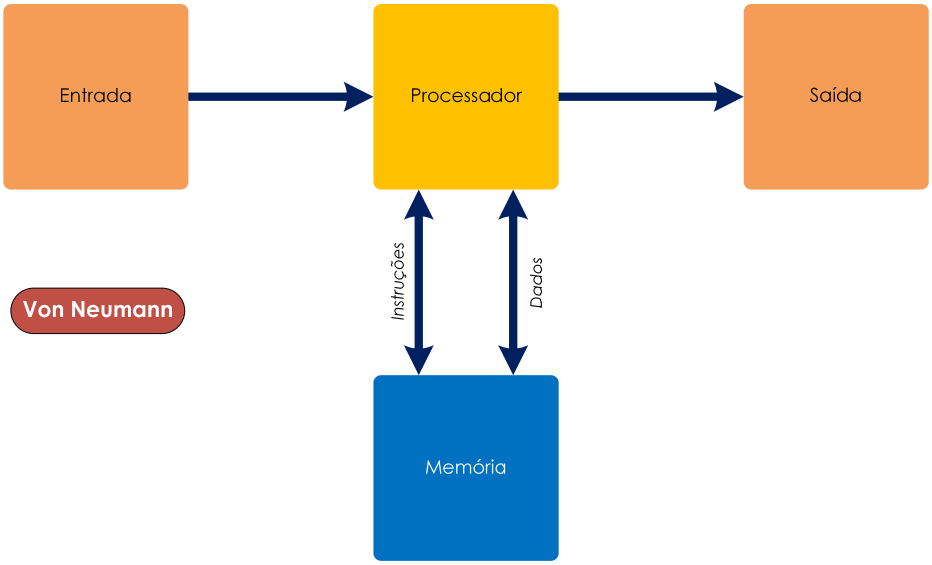


Figura 2.49 – Arquitectura do tipo von Neumann.

A arquitectura de von Neumann tem a vantagem de tornar possível que um programa altere o seu próprio código. Com o aparecimento de processadores cada vez mais rápidos, esta arquitectura

[8] Já em 1936 Alan Turing tinha publicado um artigo, intitulado "On Computable Numbers, with an Application to the Entscheidungsproblem", onde descreve um computador hipotético com uma memória infinita onde eram armazenados os dados e o próprio programa. Também em 1936 o engenheiro alemão Konrad Zuse descreveu esse conceito sem ter conhecimento do artigo de Turing .

começou a sofrer de um problema conhecido por congestionamento de von Neumann (*von Neumann bottleneck*). A necessidade de carregar tanto as instruções como os dados da mesma memória gera um tráfego muito elevado entre o processador e a memória.

A solução adoptada hoje em dia em muitos processadores é armazenar os dados e as instruções em memórias diferentes acedidas através de barramentos separados. Esta arquitectura é conhecida como Arquitectura Harvard em referência ao computador Harvard Mark I e ao seu armazenamento separado de instruções e dados (Figura 2.50). Ao contrário do Harvard Mark I em que era usada fita perfurada para guardar as instruções e contadores electromecânicos para armazenar os dados, hoje em dia é usado o mesmo tipo de memória capaz de leitura/escrita e acesso aleatório. Essas memórias podem, no entanto, ter larguras de palavras diferentes. A memória de dados tem em geral 8 bits de largura enquanto a memória de instruções tem geralmente mais bits por palavra. Outras características podem também ser diferentes como a temporização e o endereçamento.

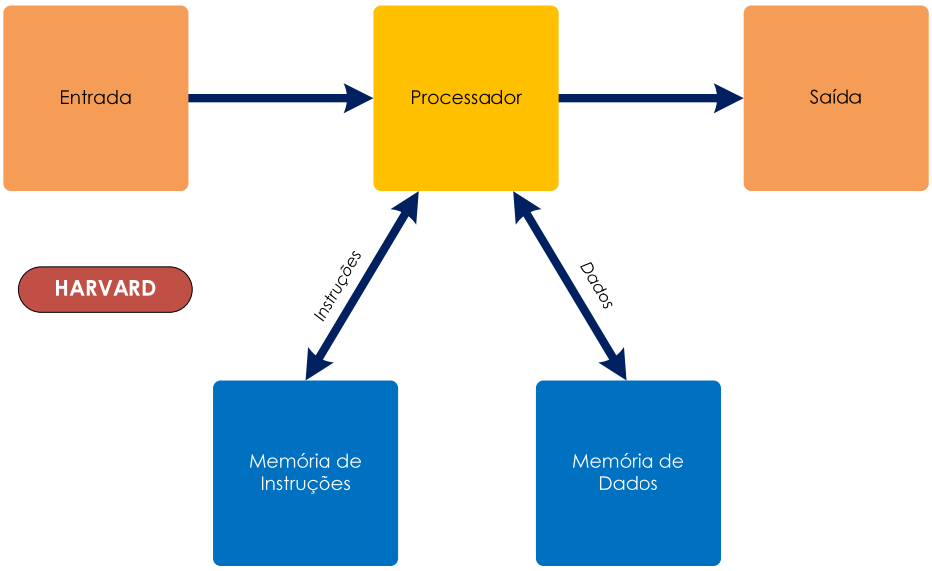


Figura 2.50 – Arquitectura do tipo Harvard.

A vantagem da arquitectura Harvard é a maior rapidez na execução do programa e as desvantagens são a maior complexidade de programação devido à existência de dois espaços de endereços separados e ao maior número de pinos que tem que ter o circuito integrado, o que dificulta o desenho das placas de circuito impresso.

Para obviar as desvantagens de cada arquitectura e usufruir das suas vantagens várias técnicas foram desenvolvidas. Uma dela tem a ver com o compromisso que há entre o tamanho da memória e a rapidez de acesso – quanto mais pequena, mais rápida. Assim sendo, em vez de se usar um único nível de memória, usam-se vários níveis. Dentro do processador é colocada uma pequena mas rápida memória (memória tampão ou *cache*) e fora do processador é colocada uma memória maior mas necessariamente mais lenta (Figura 2.51). Isto beneficia as instruções e dados mais usados que são encontrados na memória tampão não sendo portanto esperar que sejam lidos da memória principal.

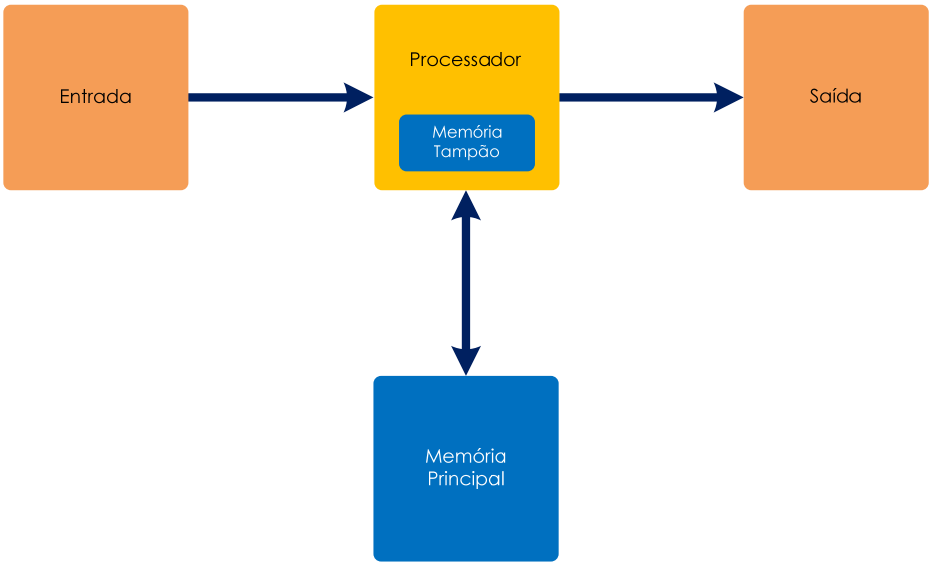


Figura 2.51 – Uso de dois níveis de memória num computador.

O uso de duas memórias tampão para as instruções e dados com barramentos separados para o processador trás os benefícios da arquitectura Harvard. Como a memória principal é a mesma o espaço de endereços e programação é a mesma da usada na arquitectura von Neumann (Figura 2.52).

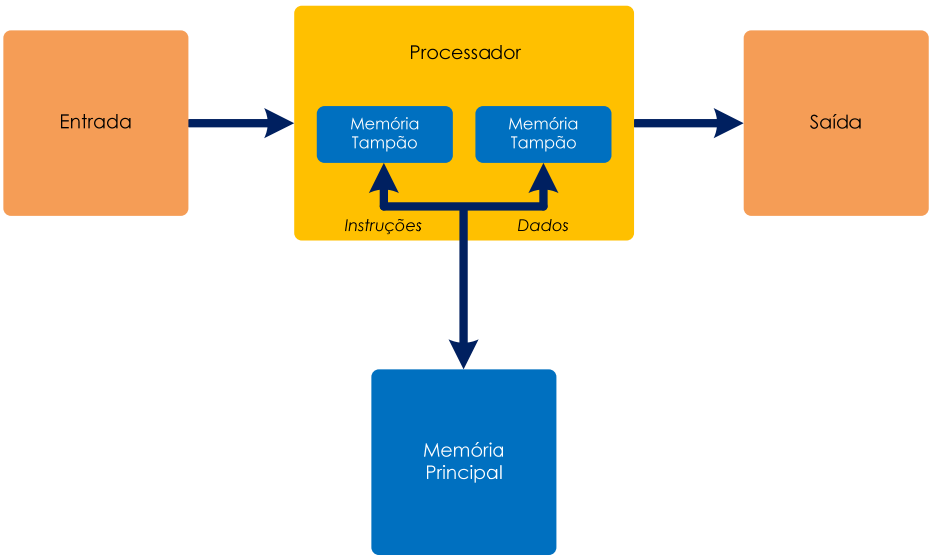


Figura 2.52 – Uso de uma arquitectura do tipo Harvard para a memória tampão e do tipo von Neumann para a memória principal.

Outra modificação, efectuada à arquitectura Harvard (*Arquitectura Harvard Modificada*), consiste em ter duas memórias principais para as instruções e dados mas permitir que a memória de instruções seja também usada como uma memória de dados. Este tipo de arquitectura encontra-se hoje unicamente em alguns casos específicos como, por exemplo, nos microcontroladores que tem tanto a memória como o processador dentro do mesmo circuito integrado. A preocupação neste caso é ter a maior velocidade de execução possível já que o sinal de relógio não é muito elevado. Outro caso onde a

arquitectura de Harvard modificada é usada é nos processadores digitais de sinal (DSP). Aí são mesmo usadas várias memórias de dados de forma a ser possível a aceder a dois (ou mais) operandos simultaneamente, aumentado a rapidez de processamento.

Quanto ao processador em si, para além de conter uma ou mais memórias tampão, também contém outros circuitos que podem ser organizados em diferentes módulos. O módulo principal contém os circuitos responsáveis por efectuar os cálculos aritméticos e lógicos. Tipicamente esse módulo é designado por ALU (*Arithmetic-Logic Unit*). Como se verá no próximo capítulo, os circuitos electrónicos digitais usados para implementar esses cálculos são circuitos combinatórios, isto é, combinações de portas lógicas (NOT, NAND, NOR, etc.) que realizam as suas funções sobre os dados colocados nas suas entradas (a saída é uma combinação das entradas). Num computador os dados sobre os quais se quer efectuar os cálculos não são sempre os mesmos. Esses circuitos combinatórios não "sabem" nada sobre como buscar dados à memória. Só "sabem" calcular o valor de saída com base nos valores presentes nas entradas em cada instante. É portanto necessário que haja outro circuito que seja responsável por ir buscar à memória os dados correctos e colocá-los na entrada da ALU, mais especificamente em registos que armazenam temporariamente os dados enquanto a ALU efectua o cálculo. Os registos são no fundo um tipo de memória. Diferem no entanto do que normalmente chamamos de memória na medida em que só armazenam um item de informação de cada vez (uma palavra) enquanto uma memória propriamente dita contém várias palavras. Uma memória tem de ser endereçada antes de poder ser lida ou escrita enquanto um registo não. O módulo responsável por ler e escrever os dados na memória e transferi-los de e para os registos internos do processador também tem outras funções como, por exemplo, a determinação de como é que uma dada instrução lida da memória é executada. Isso pode acarretar o accionamento de uma das funções da ALU (soma, subtracção, etc.) ou simplesmente a leitura de um dado de um certo sítio na memória e a sua escrita num sítio diferente. Outra das funções desse módulo de controlo é a de determinar onde se encontra a próxima instrução que deve ser executada.

O processador é portanto constituído, para além da memória tampão, pela ALU, por um conjunto de registos e pelo módulo de controlo (Figura 2.53). Um processador tem vários registos que são usados para diferentes funções. Um, dois, ou mais, são usados para fornecer os dados para a ALU, outro, normalmente designado por PC (*program counter*), armazena o endereço de memória da próxima instrução a ser executada, e muitos outros têm funções mais ou menos específicas. Ao conjunto de registos dentro de um processador é dado o nome de *ficheiro de registos*.

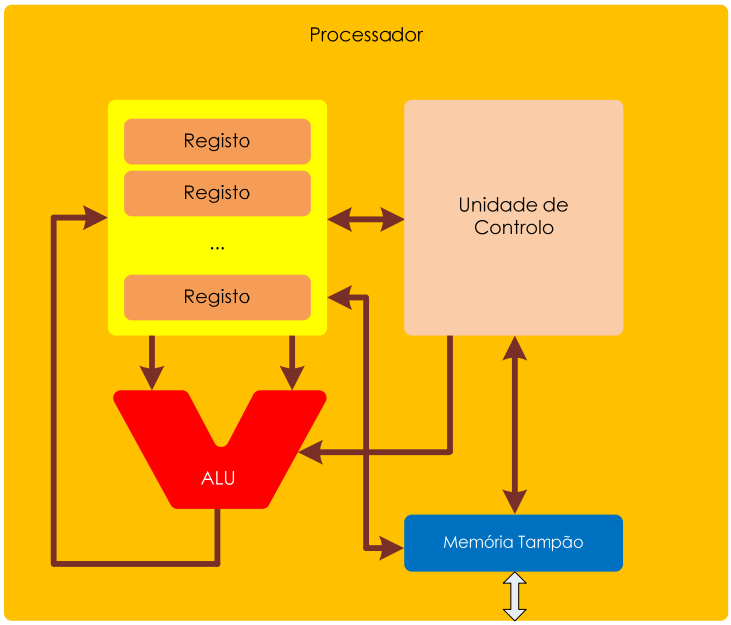


Figura 2.53 – Módulos em que se divide tipicamente um processador.

2.2.6 Fabricação de Circuitos Integrados

O domínio dos circuitos integrados na electrónica moderna deve-se não só à facilidade com que é possível construir um circuito eléctrico complexo que não tenha falhas mas principalmente à técnica de fabricação que possibilita a produção de quantidades enormes de circuitos integrados com um baixo custo por unidade. Sem isso com certeza a proliferação dos circuitos electrónicos e dos dispositivos electrónicos onde são integrados, não seria tão grande como é hoje.

Note-se, no entanto, que as fábricas em si são bastante dispendiosas, não só devido às máquinas utilizadas mas também devido à necessidade de manterem um ambiente absolutamente livre de sujidade. Qualquer partícula transportada pelo ar, por mais pequena que seja, pode potencialmente estragar um circuito integrado onde o tamanho dos transístores e outros componentes é menor do que um mícron. O custo da construção e operação da fábrica é no entanto dividido pelos biliões de circuitos integrados construídos o que torna o seu preço por unidade bastante reduzido.

A Figura 2.54 mostra o interior da fábrica de memória do tipo flash/NAND da IM Flash Technologies, uma colaboração entre a Micron e a Intel. Esta fábrica, desde Fevereiro de 2010 tem a capacidade de fabricar memórias utilizando uma tecnologia de 25 nm. Esta fábrica possui um processo produtivo totalmente automatizado. O pessoal de manutenção usa fatos completos que só deixam expostos os olhos e nariz de forma a manter extremamente limpo o ar dentro da fábrica (menos de 10 partículas por cada 10 m^3).

Figura 2.54 – Fotografia do interior da fábrica de memória do tipo flash/NAND da IM Flash Technologies uma colaboração da Micron e da Intel. Fotografia retirado da página da Internet da HotHardware.com.

A tecnologia de fabricação de um circuito integrado é hoje em dia assente na litografia e no uso do silício. Qualquer evolução que passe pela alteração do processo de fabrico é de sobremaneira dispendiosa que não é na prática implementada a não ser que traga benefícios incríveis em termos de desempenho. Veja-se, por exemplo, o caso dos circuitos ópticos. Hoje em dia estuda-se a possibilidade do uso da luz em vez do uso de electrões para a realização de microprocessadores. Com isso procura-se atingir velocidades de operação ainda mais altas que as conseguidas com os circuitos electrónicos existentes. No entanto, a não ser que o processo de fabrico de circuitos ópticos, ou optoelectrónicos seja compatível com os processos usados hoje em dia, será muito difícil que haja produtos de largo consumo competitivos com os existentes actualmente.

Criação de uma Bolacha de Silício

O primeiro passo na criação de um circuito integrado é a produção das bolachas de silício (*wafer* em inglês). Uma só bolacha, de forma circular com 300 mm de diâmetro[9] e uma espessura de 0,775 mm, irá conter muitos circuitos integrados idênticos.

O processo mais comum para criar essas bolachas é o processo de Czochralski. Esse processo permite criar cilindros de polisilício (silício cristalino) com o diâmetro desejado e com uma altura que pode chagar aos 2 m e que é depois cortado em discos. Este processo consiste nos seguintes passos:

[9] Os tamanhos utilizados variam desde os 25,4 mm até aos 450 mm. O diâmetro de 300 mm é, no entanto, o mais usado.

1) Silício com elevada pureza é derretido dentro de um cadinho[10] de quartzo. A dopagem do silício pode ser feita juntando boro ou fósforo de forma a criar silício tipo-n ou tipo-p respectivamente.
2) Uma pequena vara de polisilício é colocada dentro do cadinho. Esta vara serve de semente a partir da qual o cristal de silício é formado à medida que a sua temperatura decresce.
3) Essa vara é retirada lentamente do cadinho (1 a 10 mm por hora) ao mesmo tempo que é rodada. Durante esse processo a temperatura a que é mantido o polisilício líquido vai sendo diminuída. A velocidade de extracção e de rotação da vara, juntamente com o ritmo de descida de temperatura, determinam o diâmetro do cilindro final obtido.

O fim deste processo é um cilindro sólido de polisilício (Figura 2.55) que é cortado em discos circulares usando lâminas de diamante. Esses discos são então polidos e lavados.

Figura 2.55 – Fotografia de um cilindro de polisilício. Fonte: Wikipedia, publicada por Stahlkocher. Licença: Creative Commons Attribution ShareAlike 3.0.

Criação de uma Camada de Dióxido de Silício por Oxidação Térmica

Uma fina camada de dióxido de silício é colocada por cima do substrato de silício usando oxidação térmica. Tal é conseguido aquecendo o silício na presença de oxigénio o que pode ser feito inserindo as bolachas num forno, com uma temperatura entre os 900ºC e os 1200ºC, por onde é feito circular oxigénio como ilustrado na Figura 2.56.

[10] Recipiente que aguenta temperaturas elevadas usado para conter materiais a muito alta temperatura.

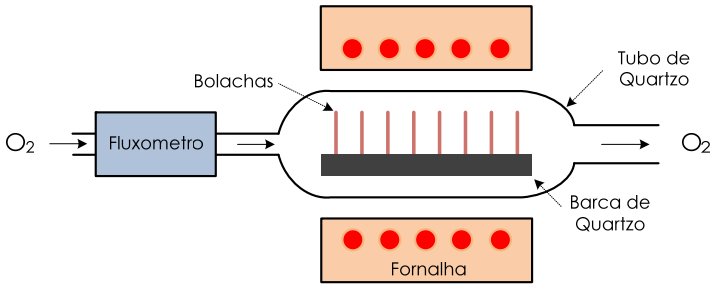


Figura 2.56 – Ilustração do processo de crescimento de uma camada de dióxido de silício em cima de uma bolacha de silício por oxidação térmica.

A reacção química que ocorre é a seguinte:

$$Si_{(s)} + O_{2(g)} \xrightarrow{900 \rightarrow 1200^{\circ}C} SiO_{2(s)}. \quad (2)$$

Em vez de oxigénio pode ser usado vapor de água com os mesmos resultados:

$$Si_{(s)} + 2H_2O_{(g)} \xrightarrow{900 \rightarrow 1200^{\circ}C} SiO_{2(s)} + 2H_{2(g)}. \quad (3)$$

A Figura 2.57 mostra uma imagem de uma camada de dióxido de silício por cima de um substrato de silício já com um padrão formado.

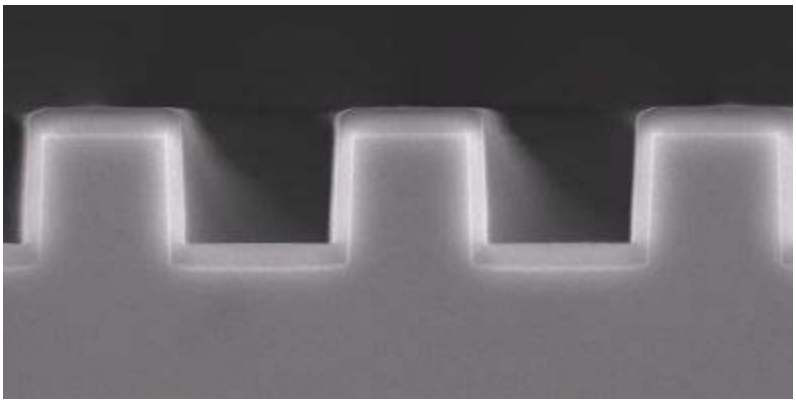


Figura 2.57 – Imagem de uma camada de dióxido de silício por cima de um substrato de silício (já com um padrão formado).

Quanto maior for o tempo que dura o processo mais alta é a camada de dióxido de silício que é criada. O uso de oxigénio ou vapor de água a alta pressão permite uma maior rapidez e a possibilidade de se trabalhar a mais baixas temperaturas. A partir da densidade de silício e do óxido de silício é possível demonstrar que uma camada de dióxido de silício com a altura *x* consome uma camada de silício com a altura 0,44*x*. Na prática conseguem-se criar camadas de dióxido de silício, utilizando este método, com uma espessura até 2 μm.

Deposição Química por Vapor

Outra técnica, que permite criar camadas de diferentes materiais inclusive de dióxido de silício ou silício policristalino é utilizando deposição química por vapor (CVD – *chemical vapour deposition*). A

estrutura, ilustrada na Figura 2.58, é semelhante à da oxidação térmica mas em que não há reacção com o material já existente na bolacha. O material que se quer depositar é criado dentro no forno na forma gasosa através da introdução dos gases apropriados e da escolha da temperatura certa.

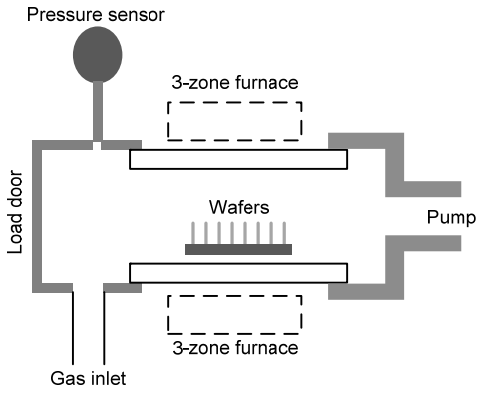


Figura 2.58 – Ilustração do processo de deposição química por vapor (CVD).

Para a deposição de uma camada de dióxido de silício são usados silano e oxigénio:

$$SiH_4 + O_2 \xrightarrow{300 \rightarrow 500^\circ C} SiO_2 + 2H_2 . \quad (4)$$

O polisilício é criado usando-se unicamente o silano a uma temperatura entre os 600 e os 650 ºC.

$$SiH_4 \xrightarrow{600 \rightarrow 650^\circ C} Si + 2H_2 . \quad (5)$$

O silano, à temperatura ambiente, existe na forma gasosa. Utilizando uma pressão entre 25 e 150 Pa consegue-se uma taxa de crescimento entre 10 e 20 nm por minuto.

Fotolitografia

A criação de padrões num material consiste em dois passos:

- Deposição e criação de um padrão numa camada preliminar;
- Transferência do padrão criado nessa camada para o material final.

No processo mais utilizado, a fotolitografia, a camada preliminar é feita de um material fotoresistivo (*photoresist*). Esse material, um polímero sensível à radiação ultra-violeta, ainda na forma líquida, é depositado sobre o substrato, o qual é feito rodar em torno do seu centro de modo a espalhar o polímero de forma uniforme e depois cozido de forma a solidificar e aderir ao material sobre o qual se encontra (Figura 2.59).

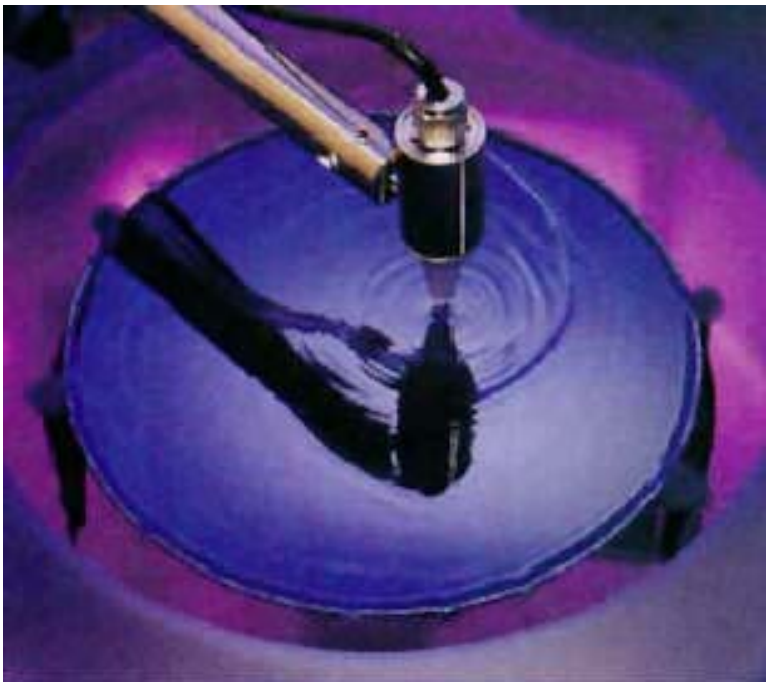


Figura 2.59 – Fotografia do processo de colocação do material fotoresistivo na bolacha.

A Figura 2.60 ilustra a primeira etapa do processo – a colocação do material fotoresistivio.

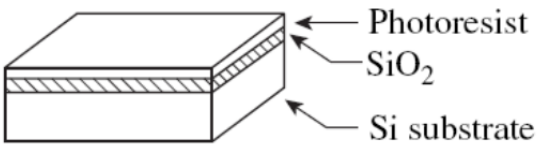


Figura 2.60 – Ilustração do polímero fotoresistivo aplicado sobre a camada de dióxido de silício na qual se quer criar um padrão.

De seguida é colocada uma máscara com o padrão desejado e é feito incidir luz ultravioleta. A luz que atravessa a máscara na zona onde não está desenhado o padrão atinge a bolacha e o material fotoresistivo alterando as suas propriedades (Figura 2.61).

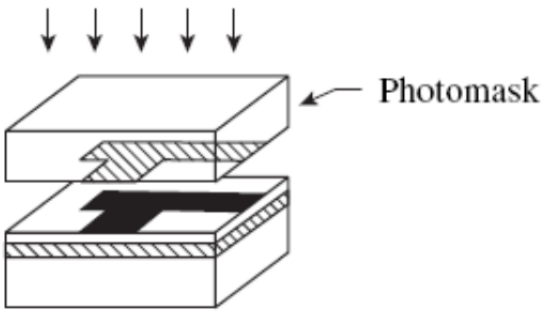


Figura 2.61 – Ilustração da colocação por cima da bolacha de uma máscara através da qual vai passar luz ultra-violeta.

A bolacha é então colocada numa solução que remove o material fotoresistivo (processo de revelação) como ilustrado na Figura 2.62.

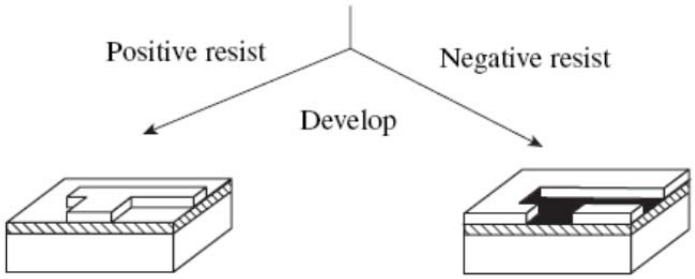


Figura 2.62 – Ilustração do efeito de revelação do material fotoresistivo para dois tipos de materiais (positivos e negativos).

Existem dois tipos de materiais fotoresistivos: os que se tornam mais solúveis pela exposição à radiação ultravioleta, chamados de "positivos"; e os que se tornam menos solúveis, chamados "negativos". Assim a utilização de materiais fotoresistivos positivos deixa na bolacha só o material fotoresistivo que estava coberta pelo padrão da máscara e no caso da utilização de matérias fotoresistivos negativos só fica o material que não estava coberto pelo padrão da máscara.

De seguida é efectuada a remoção da camada de dióxido de silício não protegida pelo material fotoresistivo (Figura 2.63) num processo designado por corrosão.

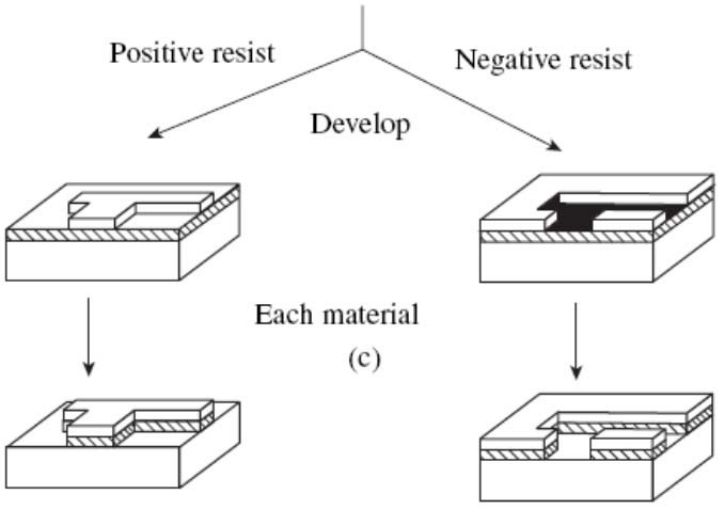


Figura 2.63 – Ilustração do processo de remoção da camada de dióxido de silício não protegida pelo material fotoresistivo.

Existem dois tipos de corrosão:

- Molhada (*wet*) – utiliza agentes químicos na fase líquida. É altamente selectiva, isto é, retira o material desejado deixando ficar os outros materiais. Pode ser isotrópica, isto é, igual em todas as direcções, ou anisotrópica.
- Seca (*dry*) – utiliza bombardeamento com iões. É anisotrópico e portanto menos susceptível a remoção lateral indesejada. Não é, no entanto, selectivo e dá-se a um ritmo mais lento que a corrosão molhada.

A Figura 2.64 apresenta alguns dos compostos usados para realizar a corrosão molhada isotrópica.

Material	Composition of the etchant	Etch rate (μm/min)
Si	$HF\ (3\,ml) + HNO_3\ (5\,ml)$	35
GaAs	$H_2SO_4\ (8\,ml) + H_2O_2\ (1\,ml) + H_2O\ (1\,ml)$	8
SiO_2	$HF\ (28\,ml) + H_2O\ (170\,ml) + NH_4F\ (113\,g)$	0.1
	$HF\ (15\,ml) + HNO_3\ (10\,ml) + H_2O\ (300\,ml)$	0.012
Si_3N_4	Buffered HF	0.005
	H_3PO_4	0.01
Al	$HNO_3\ (1\,ml) + CH_3COOH\ (4\,ml) + H_3PO_4 + H_2O\ (1\,ml)$	0.035
Au	$KI\ (4\,g) + I_2\ (1\,g) + H_2O\ (40\,ml)$	10

Figura 2.64 – Alguns materiais usados na corrosão química.

No caso de corrosão molhada isotrópica há sempre material por baixo do material que serve de máscara (material fotoresistente, por exemplo), que é removido como se ilustra na Figura 2.65. Não é portanto a técnica adequada para o fabrico de MEMS a não ser que se pretenda utilizá-la para a criação de vigas suspensas, por exemplo.

Figura 2.65 – Ilustração do efeito de uma corrosão molhada isotrópica.

Para materiais amorfos e homogéneos, a corrosão tem sempre de ser isotrópica. Em materiais cristalinos, como o silício, a corrosão pode ser anisotrópica. Isso é possível porque a taxa de corrosão é diferente consoante o plano do cristal. A Figura 2.66 mostra os diferentes planos tipicamente usados e a sua designação.

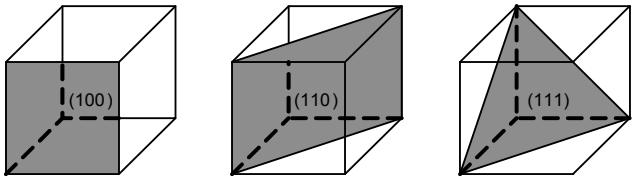


Figura 2.66 – Planos principais numa rede cúbica de silício.

Soluções com diferentes compostos têm diferentes taxas de corrosão consoante o plano. Na Figura 2.67 listam-se algumas soluções utilizadas para a corrosão do silício. É possív<el verificar, por exemplo, que relação entre as taxas de corrosão do silício pelo hidróxido de potássio (KOH) nos planos (100) e (111) é de 400. Isto significa que a corrosão dá-se principalmente segundo o plano (100) sendo a corrosão segundo o plano (111) negligenciável.

Solution	(100) Si etch rate (μm/min)	Etch rate ratio	Mask etch rate (nm/min)	Boron etch stop (cm^{-3})
KOH / H_2O 44g / 100ml (30 wt.%) @ 85°C [1]	1.4	400 for (100)/(111) 600 for (110)/(111)	3.5 (SiO_2) <0.01 (Si_3N_4)	$> 10^{20}$ rate/20
TMAH / H_2O 28g / 100ml (22 wt.%) @ 90°C [2]	1	30 for (100)/(111) 50 for (110)/(111)	0.2 (SiO_2) <0.01 (Si_3N_4)	$4 \cdot 10^{20}$ rate/40
EDP (Ethylene diamine / pyrocatechol / H_2O) 750ml / 120g / 240ml @ 115°C [3]	1.25	35 for (100)/(111)	0.5 (SiO_2) 0.1 (Si_3N_4) ≈0 (Au, Cr, Ag, Cu, Ta)	$7 \cdot 10^{19}$ rate/50

Figura 2.67 – Substâncias usadas na corrosão anisotrópica do silício.

No caso de corrosão seca são usados uma série de métodos subtractivos em que a superfície do material a corroer é removida por diferentes gases. Plasma é muitas vezes usado para aumentar a taxa de corrosão. Essa corrosão pode acontecer fisicamente através do bombardeamento com iões, quimicamente através de uma reacção química na superfície do material ou com uma combinação dos dois processos. Uma corrosão física em geral produz fronteiras mais verticais do que a corrosão química que tem a vantagem, por outro lado, de ser mais selectiva ao material que é corroído.

O último passo do processo de litografia consiste na remoção do material fotoresistente (Figura 2.68).

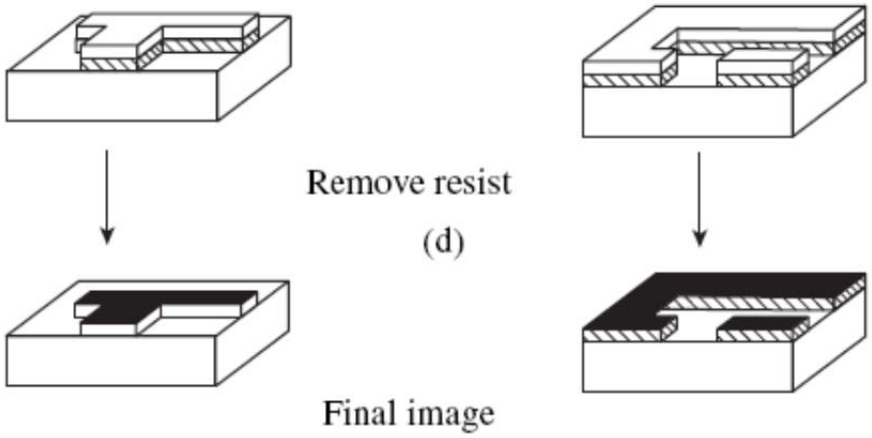


Figura 2.68 – Ilustração da remoção do material fotoresistente.

O processo de deposição, criação de padrões e transferência de padrões pode ser repetido várias vezes para criação de estruturas complexas (Figura 2.69).

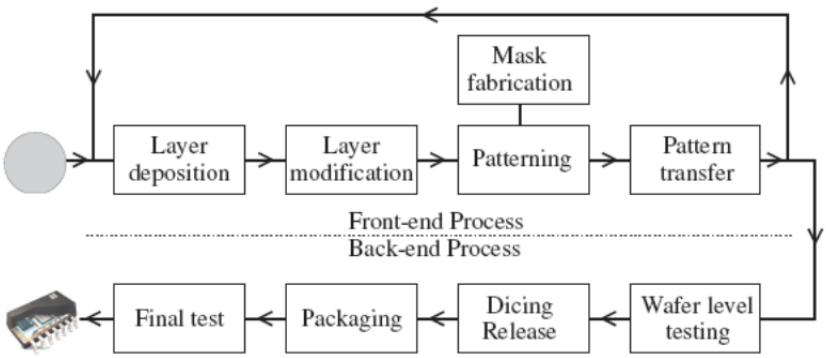


Figura 2.69 – Ilustração do processo de produção de uma MEMS.

Encapsulamento

O passo final na construção de um circuito integrado consiste em cortar a bolacha de silício de modo a separar os circuitos e coloca-los num encapsulamento que os proteja do meio ambiente e que permita a sua fácil manipulação e soldadura numa placa de circuito impresso.

Inicialmente o material usado para realizar o encapsulamento era a cerâmica sendo mais tarde substituído por plástico por ser mais barato. Há no entanto aplicações que requerem ainda o uso de encapsulamentos de cerâmica já permitem a operação numa gama de temperatura mais alargada – -55 a 125 °C comparada com a gama de -40 a 85 °C dos encapsulamentos plásticos, o que é necessário, por exemplo, em aplicações militares, perfuração e no espaço. Um dos requisitos das aplicações espaciais é o da resistência há humidade o que é conseguido usando encapsulamentos herméticos de cerâmica.

Os primeiros encapsulamentos, inventados em 1965 pela Fairchild, continham duas filas paralelas de contactos e eram chamados de encapsulamentos DIP (*dual in-line package*) embora às vezes também sejam designados por DIL. A Figura 2.70 mostra um exemplo.

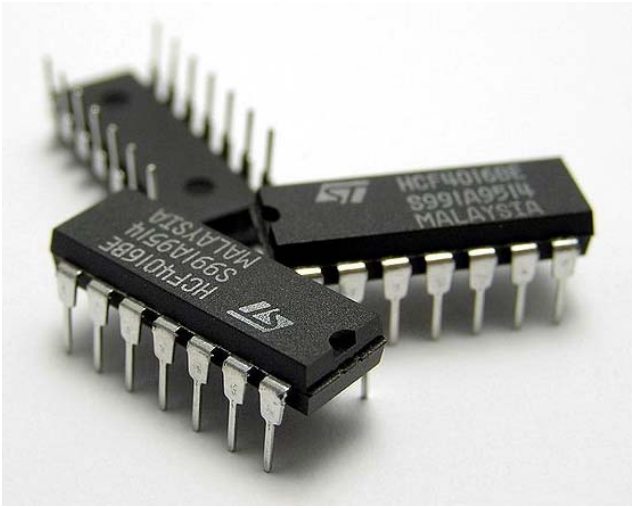


Figura 2.70 – Fotografia de circuitos integrados em encapsulamentos DIP.

Este tipo de encapsulamento é apropriado para uso em placas de circuito impresso com furos sendo os contactos soldados na parte de baixo da placa de circuito impresso. É também de fácil manuseamento por máquinas de colocação automática de componentes em placas de circuito impresso que são depois todos soldados em simultâneo em banho de solda.

A Figura 2.71 mostra o interior de um desses encapsulamentos onde se observa os fios que ligam o chip aos contactos do encapsulamento. Esses fios têm o nome, em inglês de *wire bond* (conexão por fio).

Figura 2.71 – Fotografia do encapsulamento de um microcontrolador da Intel, modelo 8742. Tirada por Ioan Sameli. Licença Creative Commons: Atribuição-Compartilhamento pela mesma Licença 2.0 Genérica.

Uma das desvantagens deste tipo de encapsulamento é o seu grande tamanho relativamente ao chip em si, o que leva a que área ocupada numa placa de circuito impresso seja muito grande.

Desde o encapsulamento DIP houve muitas evoluções com o intuito de ocupar o menor espaço possível e de permitir a existência de um maior número de pinos. É de realçar o encapsulamento SOIC (*small-outline integrated circuit*), muito usado hoje me dia (2009), que pertence a uma classe de encapsulamentos de montagem em superfície, isto é, em que a soldadura é feita do lado da placa de circuito impresso onde é assente o circuito integrado. Usando um espaçamento entre pinos menor e uma menor altura do encapsulamento é possível ter um muito maior número de circuitos integrados numa placa de circuito impresso por unidade de área (Figura 2.72).

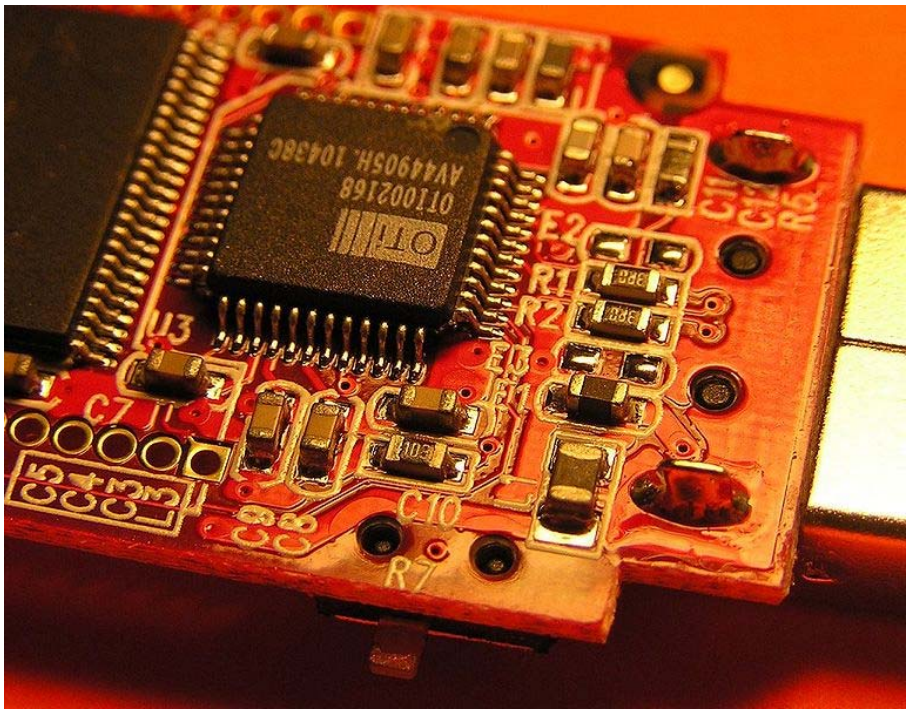


Figura 2.72 – Fotografia de uma placa de circuito impressa com vários componentes de superfície. Tirada por John Fader, Licença: Creative Commons Attribution Share Alike 3.0.

Outro encapsulamento digno de nota é o LGA (*land grid array*), usada actualmente nos microprocessadores da Intel e da AMD. Esse encapsulamento não tem pinos mas sim superfícies metalizadas que são soldados directamente na placa de circuito impresso ou colocadas em contacto com os pinos de um socket. Um exemplo encontra-se na Figura 2.73.

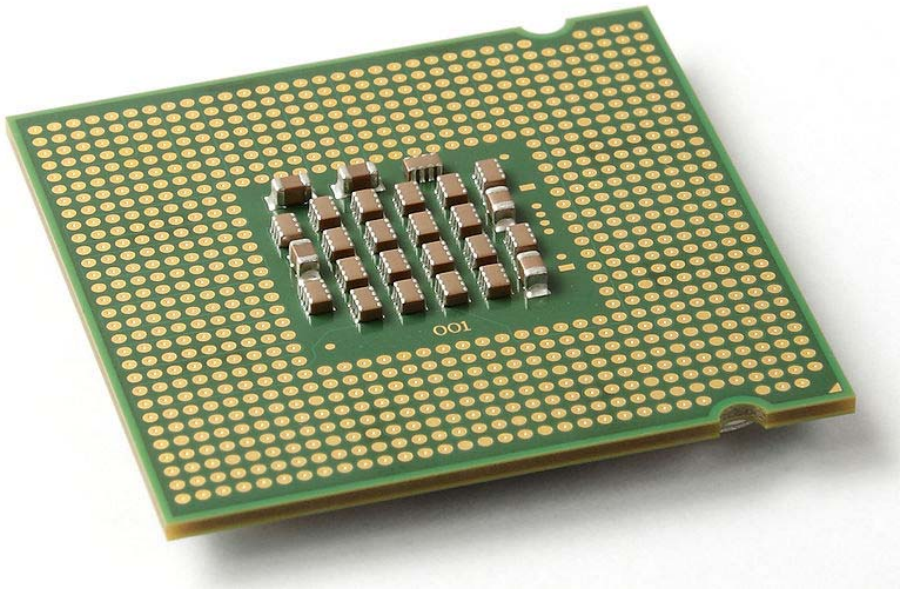


Figura 2.73 – Vista inferior de um processador Pentium 4 num encapsulamento LGA. Tirada por Eric Gaba. Licença: Creative Commons Attribution Share Alike 3.0.

A não existência de pinos leva a uma muito menor indutância nos contactos o que permite uma operação a mais alta frequência o que é bastante importante nos microprocessadores modernos. Este tipo de encapsulamento permite um muito maior número de contactos para o exterior já que usa quase toda a superfície do encapsulamento.

3. UNIDADE ARITMÉTICA E LÓGICA

O principal objectivo de um computador é efectuar cálculos matemáticos, o que hoje em dia é feito com microprocessadores e integrando circuitos electrónicos digitais. No centro dos microprocessadores encontra-se a Unidade Aritmética e Lógica (ALU), servindo os restantes componentes para suportar a tarefa de efectuar cálculos matemáticos, nomeadamente armazenando os dados que serão objecto de cálculo e o resultado do mesmo, armazenando as instruções que especificam exactamente o cálculo que é para realizar, realizando a interface com o utilizador para que este possa especificar os dados e cálculos desejados bem como visualizar os resultados, etc.

Esse circuito electrónico fulcral designado por ALU, que vem do inglês "*Arithmetic Logic Unit*" é capaz de realizar diferentes operações básicas:

- Operações aritméticas – Adição, subtracção, multiplicação, divisão e deslocamentos aritméticos de bits.
- Operações lógicas – NOT, AND, OR, XOR e deslocamentos lógicos de bits.

As operações mais completas nem sempre são realizadas por hardware, pois a área de circuito integrado que requerem pode ser demasiado elevada. Em alternativa, essas operações podem ser realizadas num circuito integrado especializado externo ao microprocessador ou por programa.

Foi John von Neumann que propôs pela primeira vez que um computador precisava de ter uma ALU, aquando do desenvolvimento do EDVAC em 1945, pois era para ele óbvio que um computador precisava de fazer cálculos matemáticos e portanto tinha que ter um circuito especializado para o efeito.

A Figura 3.1 mostra o símbolo usado normalmente para representar uma ALU.

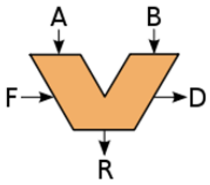


Figura 3.1 – Símbolo tipicamente usado para representar uma ALU.

De seguida apresenta-se e analisam-se diferentes algoritmos e circuitos electrónicos digitais para projectar e implementar ALUs.

3.1 Representação Inteira de Números

O projecto de uma ALU depende da forma como são representados os dados, neste caso números. Tendo em conta que esses dados tem que ser introduzidos e armazenados no computador, por outros

circuitos, como a memória ou o barramento de entrada, é natural que seja adoptada uma mesma representação nos vários componentes de um computador. Essa representação, usando "0" e "1", baseia-se no sistema de numeração binário e é hoje usada universalmente. Em termos electrónicos esses dois símbolos são representados usando dois níveis de tensão ou corrente distintos. Isto permite uma maior fiabilidade e imunidade ao ruído aquando da sua propagação através de circuitos electrónicos e meios de comunicação. Em termos tecnológicos é também mais fácil armazenar a informação utilizando só dois símbolos distintos.

Qualquer número é portanto representado num computador usando base 2. Claro que em termos teóricos poderiam ser usadas outros sistemas de numeração posicionais com base diferente de 2, mas a simplicidade de representação e a facilidade de cálculo com bases que são potências de 2 tornaram o uso destes sistemas de numeração, em computador, universal.

Um número binário com N algarismos a_i é dado por

$$x = \sum_{i=0}^{N-1} a_i 2^i \ . \tag{5}$$

O bit a_0 é portanto o bit menos significativo (LSB) e o bit a_{N-1} é o bit mais significativo (MSB).

A Tabela 3.1 mostra os 16 números inteiros positivos que são possíveis de representar com 4 bits.

Tabela 3.1 – Representação binária de números inteiros positivos. Exemplo com 4 bits.

Decimal	Binário			
	MSB			LSB
15	1	1	1	1
14	1	1	1	0
13	1	1	0	1
12	1	1	0	0
11	1	0	1	1
10	1	0	1	0
9	1	0	0	1
8	1	0	0	0
7	0	1	1	1
6	0	1	1	0
5	0	1	0	1
4	0	1	0	0
3	0	0	1	1
2	0	0	1	0
1	0	0	0	1
0	0	0	0	0

No caso de números inteiros negativos existem diversas formas de os representar:

- Complemento para dois.
- Complemento para um.
- Sinal/magnitude.

A escolha de uma destas representações tem implicações nas ALUs projectadas levando a circuitos diferentes. Com o seu desenvolvimento veio a verificar-se que o uso do complemento para dois trazia benefícios em termos da relação custo/desempenho. Isso deve-se fundamentalmente ao facto que com a representação de complemento para dois não é preciso ter circuitos individualizados para somar e para subtrair. Basta ter um para somar e outro para calcular o complemento para 2 o que é muito fácil – basta trocar os "0" com os "1" e somar "1". A Tabela 3.2 apresenta alguns exemplos de números inteiros representados em binário com 4 bits usando complemento para 2. Observe-se, por exemplo como o simétrico de 7 ("0111") obtém-se trocando o valor dos bits ("1000") e somando 1 ("1001"). Querendo realizar a operação "7 - 1", por exemplo, basta somar 7 ("0111") a -1 ("1111") o que dá 6 ("0110").

Apesar de, ao contrário do complemento para um, não ser necessário detectar a passagem por zero, verifica-se na Tabela 3.2 que o sistema de numeração binário em complemento para dois não é simétrico existindo para *N* bits a representação do número -2^{N-1} mas não a representação do número $+2^{N-1}$ (11…10 + 1 = 10…00).

Tabela 3.2 – Exemplos de números inteiros binários usando complemento para 2.

Decimal	3.1.1.1.1.1		Binário	
	3.1.1.1.1.2 3.1.1.1.1.3			LSB
7	0	1	1	1
6	0	1	1	0
5	0	1	0	1
4	0	1	0	0
3	0	0	1	1
2	0	0	1	0
1	0	0	0	1
0	0	0	0	0
-1	1	1	1	1
-2	1	1	1	0
-3	1	1	0	1
-4	1	1	0	0
-5	1	0	1	1
-6	1	0	1	0
-7	1	0	0	1

-8	1	0	0	0

Na Figura 3.2 mostra-se os números decimais correspondente a uma representação binária com 4 bits. No caso do uso de uma representação sem sinal as 16 (2^4) combinações possível com 4 bits são usadas para representar os números decimais desde o 0 ao 15. Se os mesmos 4 bits forem interpretados como formando uma representação em complemento para dois então o bit mais significativo representa o sinal algébrico e os outros 3 bits são usados para representar 8 (2^3) números decimais. Temos assim 16 combinações possíveis: uma para representar o 0, sete para representar os números positivos de 1 a 7 e oito para representar os números negativos de -8 a -1. Há portanto mais números negativos representados do que positivos (o número -8 é representado mas o número +8 não é).

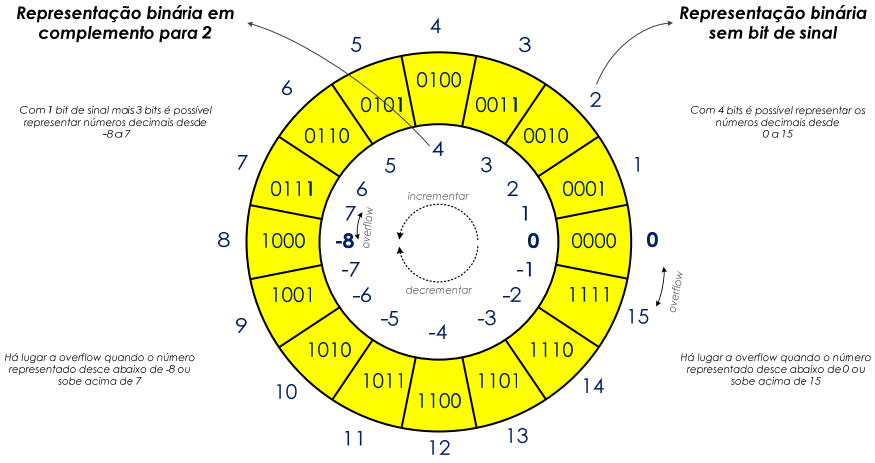


Figura 3.2 – Ilustração da representação de números em formato binário sem sinal e em formato binário em complemento para 2. Exemplo para palavras com 4 bits.

3.2 Adição

A adição é a operação aritmética mais simples e está na base de todas as outras operações aritméticas. Existem diversos circuitos capazes de realizar a soma de dois números existindo um compromisso entre tempo de processamento, área ocupada e consumo energético. As características destes diferentes circuitos podem ainda ser optimizadas tendo em conta a tecnologia alvo.

A soma de dois números, efectuada com papel e lápis, consiste em alinhar os dois números à direita e somar, algarismo a algarismo, a começar pela direita. Caso a soma seja maior do que 9 temos que somar um "1" extra à soma de algarismos seguinte ("e vai 1"). Diz-se assim haver lugar ao transporte de um "1" (*carry* em inglês). No caso dos computadores os números têm todos o mesmo número de bits e portanto não é preciso alinhá-los. Como é usada base 2, "vai 1" se os dois algarismos forem "1" (ou se um deles for "1" e o bit de transporte também for "1"). A Figura 3.3 mostra um exemplo (10+3 = 13).

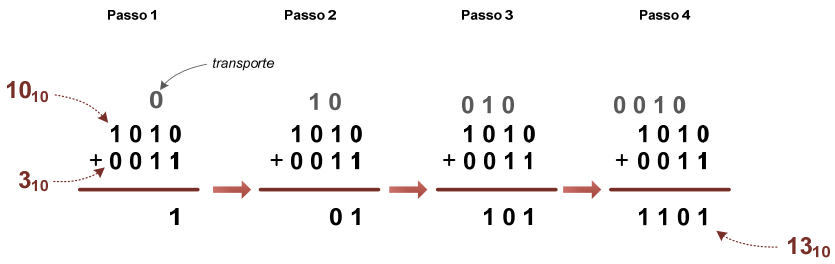


Figura 3.3 – Ilustração da soma de dois números em base 2 (10 + 3 = 13).

A soma de dois números positivos pode dar origem a overflow, isto é, o resultado pode ser demasiado grande para ser representado com o mesmo número de bits dos operandos. A Figura 3.4 apresenta um exemplo. A soma de10 com 12 devia dar 22 mas esse número é demasiado grande para ser representado com 4 bits (o máximo é 15). O valor obtido é 6 (0110_2) que é o que se obtém removendo o bit mais significativo de 22 representado com 5 bits (10110_2).

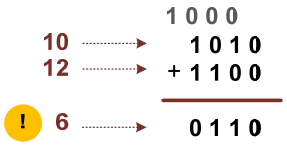


Figura 3.4 – Ilustração da soma de dois números representados com 4 bits sem sinal (10 + 12). O resultado obtido está errado devido a overflow.

A Figura 3.5 ilustra a relação entre os números envolvidos na soma de 10 com 12 no caso de uma representação binária sem sinal com 4 bits. O resultado obtido é 6 em vez de 22 devido ao valor máximo passível de se representar com 4 bits (sem sinal) é 15 e portanto menor do que o resultado correcto.

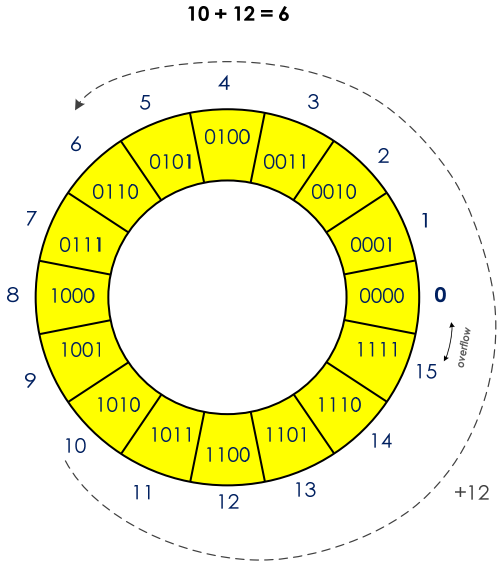


Figura 3.5 – Ilustração da soma de 10 com 12 usando uma representação de números em formato binário sem sinal com 4 bits. Devido a overflow o resultado é errado (6 em vez de 22).

A Figura 3.6 apresenta todos os resultados possíveis da soma de dois números entre 0 e 15. Esses números são os que se podem representar com 4 bits numa notação binária sem sinal. Se o resultado tiver que ser representado também com 4 bits então não pode ser maior do que 15. Em todos os casos em que isso acontece o resultado armazenado será errado fruto da eliminação do 5º bit que seria necessário para representar, por exemplo, o número 30. Esses resultados incorrectos estão assinalados a cor-de-laranja no lado direito da Figura 3.6.

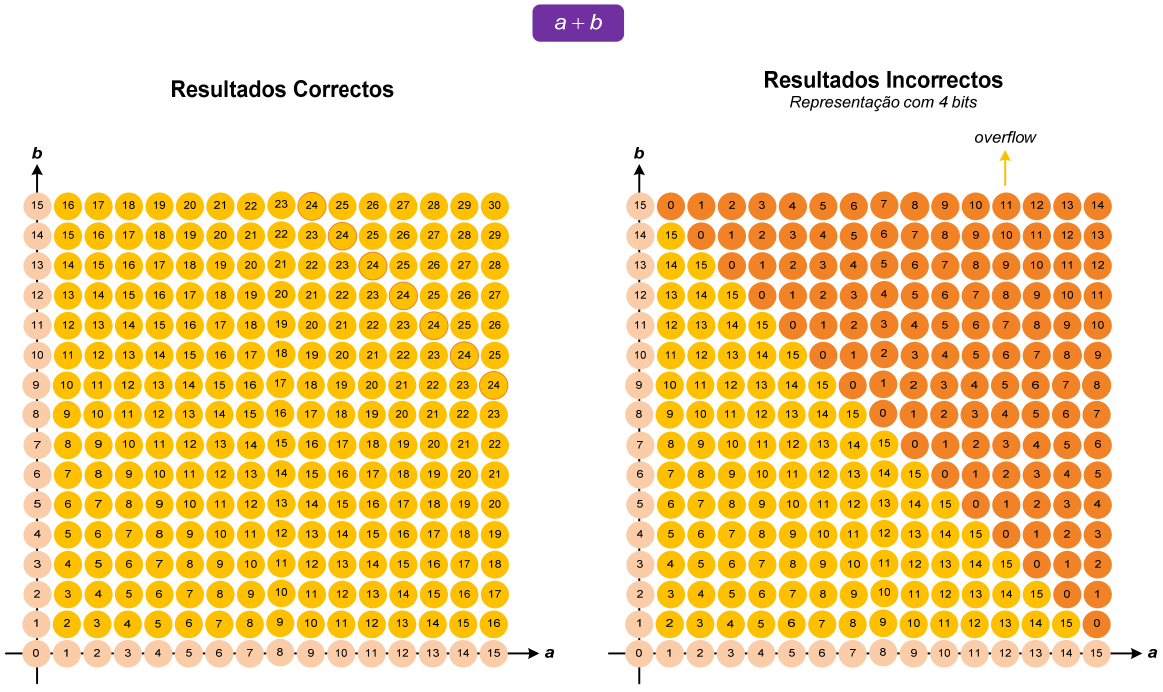


Figura 3.6 – Resultados obtidos na soma de dois números representados com 4 bits em complemento para 2. O resultados marcados a cor-de-laranja estão errados e são fruto de overflow.

A indicação da ocorrência de overflow pode ser obtida a partir do valor do último bit de transporte. No exemplo da Figura 3.4, por exemplo, verifica-se a ocorrência de overflow dado que o último bit de transporte vale 1.

Se os números forem representados em complemento para dois então o procedimento é o mesmo caso os operandos sejam positivos ou negativos. A Figura 3.7 mostra o exemplo da soma de -6 com 3. Note-se que os bits dos operandos e do resultado são exactamente os mesmos do que no exemplo apresentado na Figura 3.3. O bit mais significativo vale "0" no caso de números positivos e "1" no caso de números negativos.

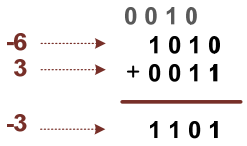


Figura 3.7 – Ilustração da soma de dois números representados em complemento para 2 (-6 + 3 = -3).

Enquanto que na representação binária com 4 bits de números sem sinal algébrico é possível representar o números de 0 a 15, no caso da representação em complemento para dois os números representáveis com os mesmos 4 bits vão de -8 a 7.

Pode acontecer que o resultado da soma não seja possível de ser representado com o mesmo número de bits dos operandos. No caso de somadores de 4 bits para números representados em complemento para 2 a soma de 4 com 5 dá origem a 9 está fora da gama de números representáveis (-8 a 7). Diz-se que neste caso ocorreu um overflow. Ao efectuar-se a soma tal como anteriormente obtém-se o resultado de -7.

Figura 3.8 – Ilustração da soma de dois números representados em complemento para 2 (4 + 5) com 4 bits quando o resultado (9) não é passível de ser representados com os mesmos 4 bits.

A Figura 3.9 apresenta todos os resultados possíveis para a soma de dois números entre -8 e 7 (lado esquerdo). No caso do resultado ter de ser representado com 4 bits usando uma notação em complemento para 2, existem diversos casos em que se observa overflow (lado direito).

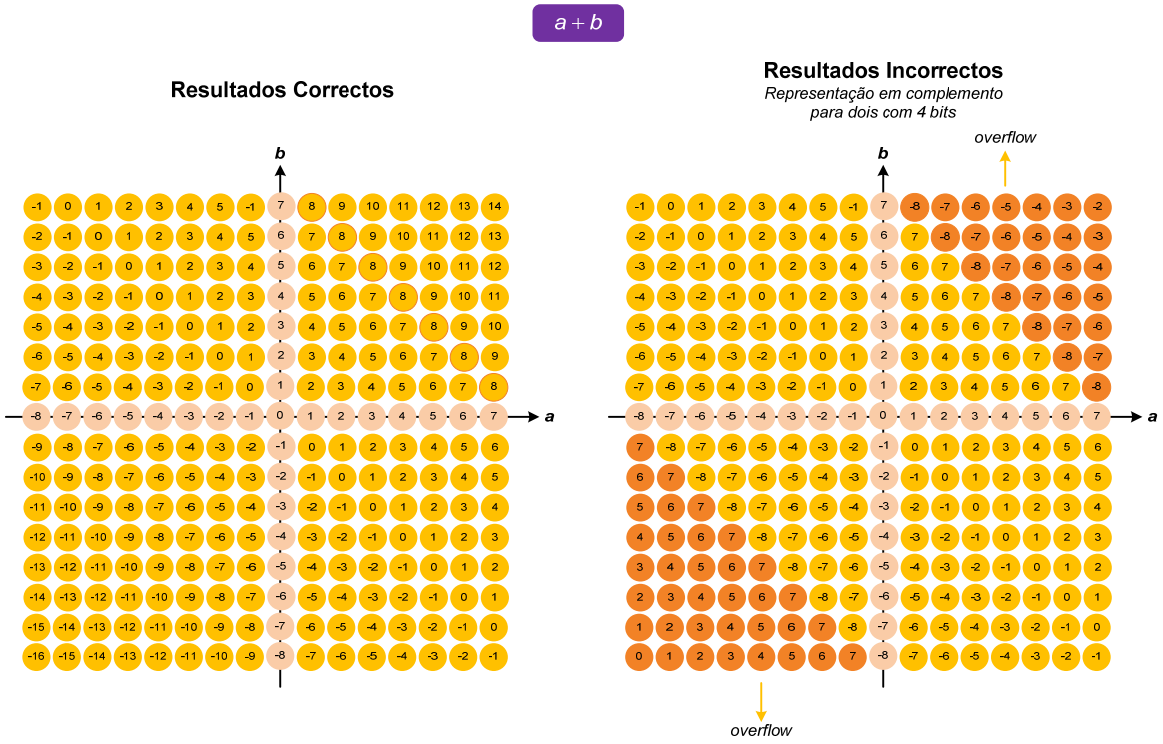


Figura 3.9 – Resultados obtidos na soma de dois números representados com 4 bits em complemento para 2. O resultados marcados a cor-de-laranja estão errados e são fruto de overflow.

No caso da soma de números com sinal a situação de overflow acontece quando os dois últimos bits de transporte são diferentes. No exemplo da Figura 3.8, onde se soma 4 com 5 verifica-se que o penúltimo bit de transporte é 1 e o último é 0. Como o soma dos bits mais significativos não deu origem a um bit de transporte quer dizer que o bit de transporte anterior deu origem a uma soma igual a 1 ou seja uma soma negativa. Os dois bits mais significativos dos operandos tinham que ser zero caso contrário haveria lugar a um bit de transporte depois de somados os últimos bits. A soma de dois números positivos deu origem a um número negativo, ou seja a um overflow.

A Figura 3.10 apresenta-se outro exemplo. Neste caso a soma de -3 com -8 neste caso os dois últimos bits de transporte também são diferentes.

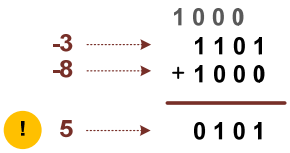


Figura 3.10 – Ilustração da soma de dois números representados em complemento para 2 (-3 + (-8)) com 4 bits quando o resultado (-12) não é passível de ser representados com os mesmos 4 bits.

Como o penúltimo bit de transporte é 0 e o último bit de transporte é 1 isso significa que os dois bits de sinal dos operandos são 1. A soma desse dois "1" dá origem a 0 e "vai 1". O resultado da soma é assim positivo. Tinha-se então dois números negativos que somados deram resultado a um número positivo o que indica a ocorrência de overflow.

3.2.1 Somador Ripple-Carry

A forma mais directa de criar um circuito para somar dois números é replicar o procedimento seguido por nós quando somamos com papel e lápis.

A Figura 3.11 mostra o diagrama de blocos de um circuito para realizar este procedimentos. Cada operação de soma de dois bits é efectuada por um circuito representado pelo bloco FA (somador completo, *Full Adder*), cujas entradas são os bits das palavras a somar (a_i e b_i) e o bit de transporte da soma anterior (c_i). Esses blocos têm duas saídas: o resultado da soma (s_i) e o bit de transporte resultante da soma (c_{i+1}).

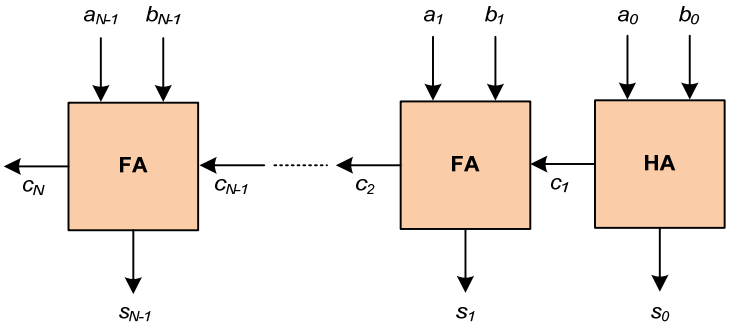


Figura 3.11 – Diagrama de blocos do circuito somador do tipo "Ripple-Carry".

No caso da soma dos bits mais à direita (primeira soma) não há nenhum transporte a entrar. O circuito somador de bits neste caso é chamado de meio-somador, ou em inglês, *half-adder* (bloco HA).

A tabela de verdade do meio-somador é a representada na Tabela 3.3.

Tabela 3.3 – Tabela de verdade do meio-somador (HA).

Entradas		Saídas	
a_i	b_i	c_{i+1}	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Matematicamente as saídas do bloco HA são dadas por

$$\begin{aligned} s_i &= a_i' b_i + a_i b_i' = a_i \oplus b_i \\ c_{i+1} &= a_i \cdot b_i \end{aligned} \quad . \qquad (1)$$

Estas funções lógicas são implementadas com o circuito lógico da Figura 3.12.

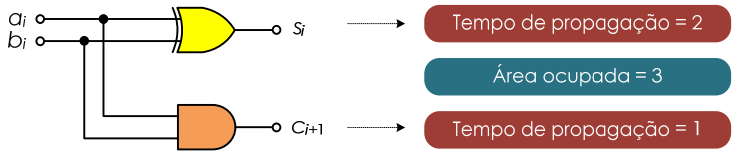


Figura 3.12 – Circuito digital de um meio somador.

Para avaliar a eficiência dos circuitos digitais há dois parâmetros que devem ser consideradas em primeira análise: a área ocupada e o tempo de propagação, isto é, o tempo que leva a saída do circuito a reflectir uma mudança das suas entradas. Para além deste dois parâmetros mais importantes, o consumo de potência tem vindo a ter uma importância crescente na caracterização dos circuitos. O tempo de propagação depende não só do tipo de porta lógica como do número de entradas e número de saídas (número de outras portas que podem ser ligados à saída de uma dada porta) e da tecnologia utilizadas para implementar a porta lógica. Uma forma simplificada de fazer a contabilização do tempo de cálculo de um circuito lógico bem como da área ocupada é usar um modelo simplificado como o proposto por Akhilesh Tyagi em [3]. Esse modelo considera que todas as portas lógicas básicas têm um tempo de propagação unitário com excepção da porta XOR, quando implementada em CMOS, que tem o dobro do tempo de propagação e o dobro da área ocupada. A presença de inversores, buffers e interligações não é contabilizada nem para o tempo nem para a área. A Tabela 3.4 resume esse modelo simplificado.

Tabela 3.4 – Modelo teórico para calcular o tempo de propagação e área ocupada por diferentes portas lógicas que é independente da tecnologia alvo, segundo [3].

Célula	Símbolo	Área	Tempo de Propagação
Inversor/Buffer	INV, BUFF	0	0
Interligação	-	-	-
Portas básicas de 2 entradas	AND, NAND, OR, NOR	1	1
Portas compostas de 2 entradas	XOR, XNOR	2	2
Portas de *m* entradas	ANDm, NANDm, ORm, NORm, XORm, XNORm	*m*-1	$\lceil \log_2(m) \rceil$

No caso do circuito da Figura 3.12 existe uma porta XOR e uma porta AND o que corresponde a uma área total de 3. O tempo de propagação para a saída da soma é determinado pela porta XOR e é portanto 2 enquanto o tempo de propagação para a saída do bit de transporte é determinado pela porta AND e portanto vale 1.

A função lógica entre as entradas e saídas do bloco FA pode ser expressa através da seguinte Tabela de Verdade. Note-se que $c_0 = 0$.

Tabela 3.5 – Tabela de Verdade do somador completo.

Entradas			Saídas	
a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A saída s_i do somador completo pode ser obtida de (1) usando 3 variáveis.

$$s_i = (a_i \oplus b_i) \oplus c_i \tag{2}$$

A saída c_{i+1} pode ser obtida com um mapa de Karnaugh como o representado na Figura 3.13.

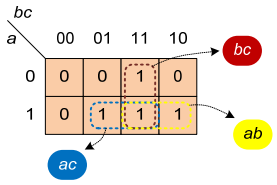


Figura 3.13 – Mapas de Karnaugh para o problema proposto.

A expressão matemática é portanto

$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i \,. \tag{3}$$

Esta expressão pode ser escrita de outra forma colocando a variável c_i em evidência:

$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i + b_i) \,. \tag{4}$$

Esta função lógica pode ser implementada por um conjunto de portas lógicas interligadas conforme se ilustra na Figura 3.14.

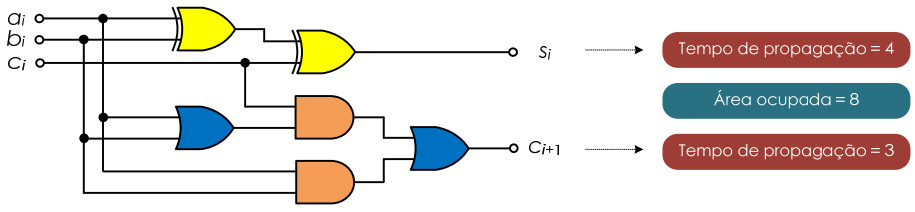


Figura 3.14 – Realização lógica do somador completo (bloco FA).

No caso da Figura 3.14 o circuito que calcula o bit de soma é constituído por duas portas XOR o que dá uma área ocupada de 4 e um tempo de propagação de 4. Já o circuito que efectua o cálculo do bit de transporte é constituído por duas portas OR e duas portas AND que operam em paralelo. Assim a área ocupada é de 4 e o tempo de propagação é de 3. No total a área ocupada pelo somador completo tal como é implementado na Figura 3.14 é de 8.

Em alternativa ao circuito da Figura 3.14 pode-se construir um circuito que tem uma área ocupada menor à custa de um maior tempo de propagação para a saída do bit de transporte. Isso consegue-se observando que a equação (4) pode também ser escrita como

$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i \oplus b_i). \tag{5}$$

A única diferença entre as duas equações acontece no termo entre parêntesis. Só quando a_i e b_i valem 1 é que esse termo é diferente. No entanto nesse caso o termo $a_i b_i$ será 1 o que faz com que c_{i+1} seja 1 independentemente do termo entre parêntesis. A equação (5) permite que se reutilize parte do circuito eléctrico usado para o cálculo da soma, nomeadamente o ou-exclusivo entre a_i e b_i que está presente em (2). Obtém-se assim o circuito da Figura 3.15.

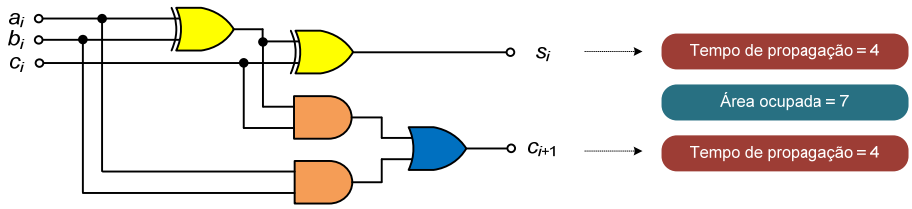


Figura 3.15 – Realização lógica alternativa do somador completo (bloco FA).

Estes circuitos digitais podem ser implementados com transístores substituindo cada porta lógica pela sua implementação respectiva. Existem no entanto outros circuitos possíveis que dão lugar ao mesmo resultado. Um desses exemplos é o da Figura 3.16.

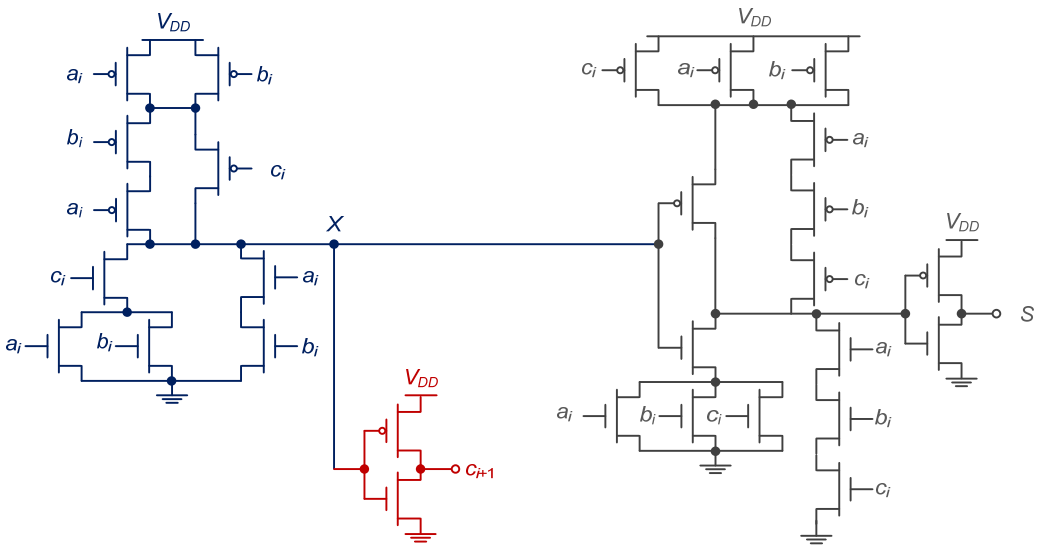


Figura 3.16 – Exemplo de implementação em CMOS de um somador completo.

Veja-se, por exemplo, a parte inferior do circuito que dá origem à variável x_i, que é a negação do bit de transporte de saída do circuito. Essa parte é constituída por dois ramos em paralelo o que equivale em termos lógicos a uma soma e em termos booleanos a um OR. O ramo mais à esquerda tem o transístor comandado por c_i ligado em série com o paralelo de dois transístores comandados por a_i e b_i. A expressão matemática é portanto $c_i \cdot (a_i + b_i)$. O ramo da direita contém dois transístores em série comandados por a_i e b_i o que representa $a_i b_i$. Os dois ramos em conjunto implementam a função lógica

$$X' = a_i \cdot b_i + c_i \cdot (a_i + b_i). \tag{6}$$

Como se trata da parte inferior do circuito CMOS deve ser encarado como funcionando em lógica negativa. Assim sendo o valor lógico de X é a negação de (6). O bit de transporte de saída é a negação de X o que corresponde a (6) que é o mesmo que (4). A parte superior do circuito implementa a mesma função, mas em lógica complementar, ou seja, positiva.

O mesmo tipo de raciocínio pode ser aplicado ao resto do circuito da Figura 3.16 dando origem à equação (2).

O menor tempo de propagação para o circuito do somador Ripple-Carry consegue-se implementado os somadores completos de um bit como os da Figura 3.14. Nesse caso, o tempo de propagação até à saída do último bit de transporte é de 1+3(*N*-1) correspondente um meio somador e a *N*-1 somadores completos. O tempo de propagação do último bit de soma é semelhante com a excepção do tempo de propagação através do último somador completo que é de 4 em vez de 3. Tem-se assim um tempo de propagação para a soma de 1+3(*N*-2)+4 (Figura 3.17).

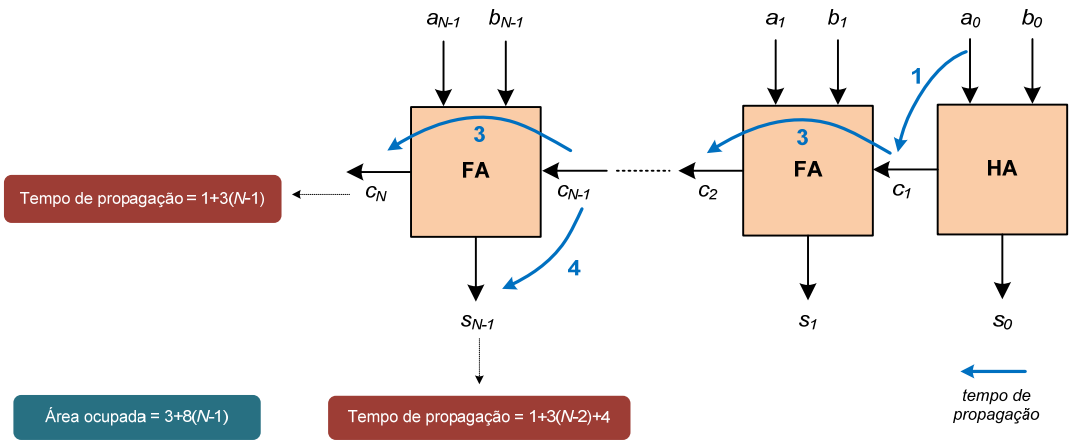


Figura 3.17 – Diagrama de blocos do circuito somador do tipo "Ripple-Carry".

A área ocupada é de 3+8(*N*-1), 3 devido ao meio somador e 8 devido a cada um dos somadores completos.

O somador Ripple-Carry tem a vantagem de ser simples mas a desvantagem de ser lento e do tempo de propagação aumentar proporcionalmente com o número de bits das palavras a somar. Cada bloco FA só pode efectuar o cálculo após o bloco anterior ter concluído, devido ao bit de transporte.

O bit de transporte de saída do somador é o originado no último bloco FA (c_N). Esse bit pode ser usado, por exemplo, para ligação a outro somador do mesmo tipo por forma a aumentar o número de bits dos operandos e resultado.

Como se viu anteriormente, no caso de soma de números positivos a condição de overflow pode ser detectada usando o último bit de transporte (Figura 3.18).

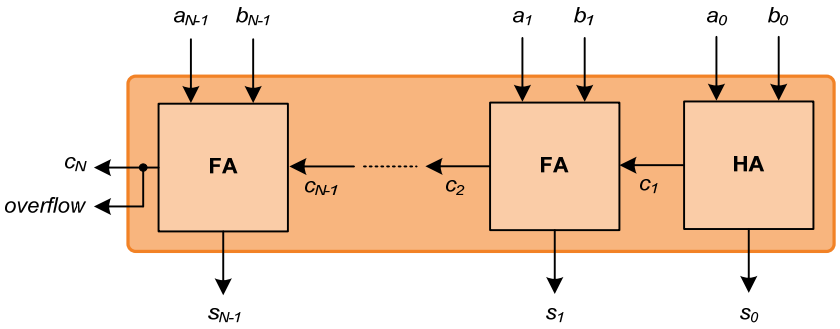


Figura 3.18 – Diagrama de blocos do circuito somador do tipo "Ripple-Carry" com indicação de overflow no caso de números sem sinal.

No caso de número com sinal o somador é exactamente o mesmo, caso sejam representados em complemento para dois, no entanto o bit de overflow é determinado de forma diferente. Neste é determinado realizando-se o ou-exclusivo dos dois últimos bits de transporte (Figura 3.19).

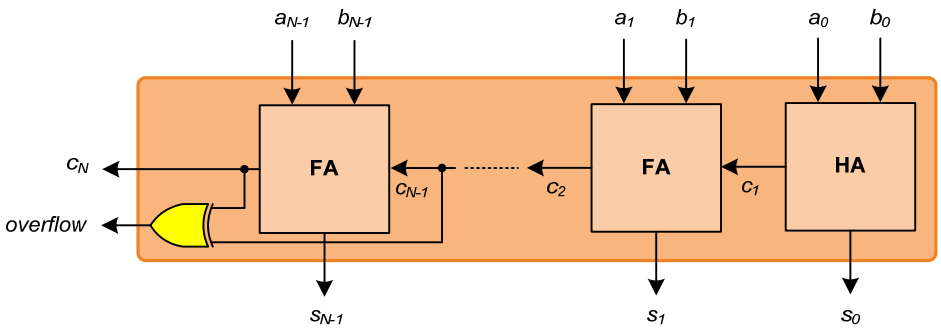


Figura 3.19 – Diagrama de blocos do circuito somador do tipo "Ripple-Carry" com indicação de overflow no caso de números com sinal.

3.2.2 Somador Carry-Select

O somador Ripple-Carry é de simples concepção mas é lento, pois a soma de cada bit tem de esperar pela soma do bit anterior. Isso acontece porque é necessário esperar que o transporte do bit anterior seja conhecido. Uma solução para tornar o somador mais rápido é dividir a palavra a meio e realizar a soma das duas metades em paralelo. Como a soma da metade esquerda depende do transporte da metade direita, o que o somador Carry-Select faz é efectuar dois cálculos da metade esquerda, um presumindo que o transporte que vem da direita é 0 e outro que é 1. Depois de completadas as 3 somas então o somador selecciona, usando um multiplexer, qual das duas somas pré-calculadas para os bits mais significativos deve usar tendo em conta o transporte da metade direita. A Figura 3.20 mostra o diagrama de blocos desse somador para um caso concreto da soma de palavras de 8 bits e uso de 2 andares (divisão das palavras em duas metades).

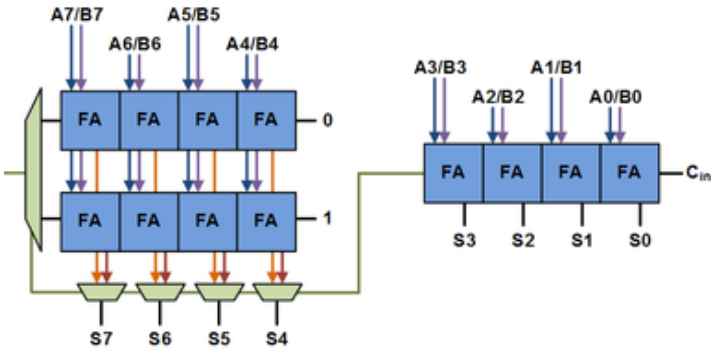


Figura 3.20 – Diagrama de blocos de um somador transporte Select de 8 bits e dois andares.

O tempo que demora a efectuar o cálculo de operandos de *N* bits é portanto o tempo de cálculo de 1 somador de *N*/2 bits completos mais o tempo de propagação de um multiplexer. A desvantagem, em relação ao Ripple-Carry é que requer mais área para ser implementado.

A Figura 3.21 mostra a arquitectura de um somador Carry-Select com *k* andares, sendo cada andar responsável pela soma de *w*/*k* bits, em que *w* é o número de bits da palavra a somar.

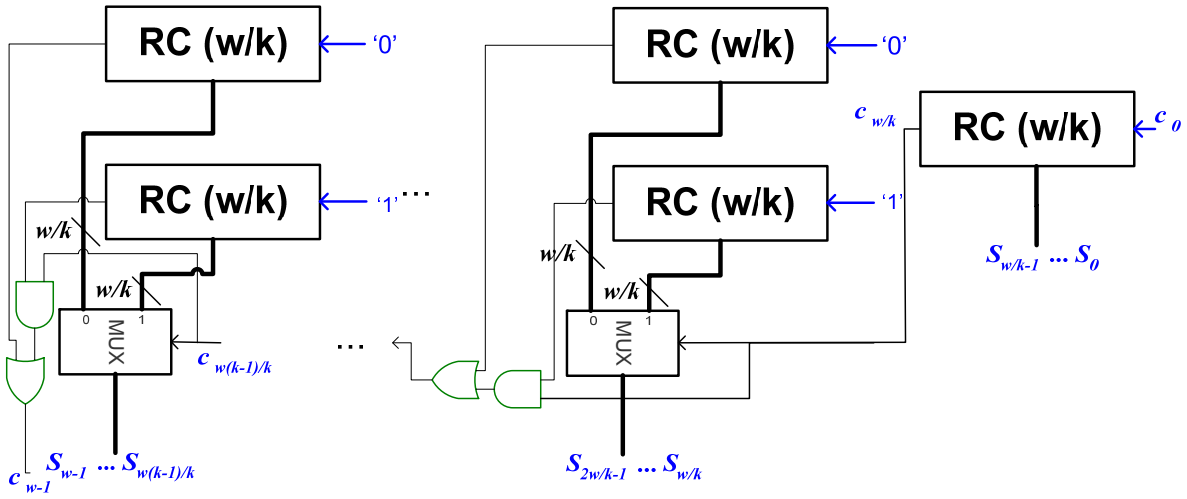


Figura 3.21 – Arquitectura de um somador transporte Select de *k* andares.

Cada andar tem dois somadores Ripple-Carry com *w*/*k* bits, com excepção do primeiro andar, o mais à direita, que tem só 1 somador Ripple-Carry pois o valor do transporte de entrada desse andar é conhecido (é zero).

3.2.3 Somador Carry-Lookahead

A rapidez com que é possível somar dois números está sempre limitada pelo transporte que se propaga de bit para bit. O somador Carry-Lookahead permite melhorar a rapidez da soma prevendo se determinado bit, ou conjunto de bits, vai dar origem a um transporte nos bits seguintes.

Existem três casos que podem ser distinguidos (Figura 3.22):

- Aniquilação – Se os dois bits a somar forem 0, o bit de transporte é sempre 0 independentemente do transporte que vem do lado direito.
- Geração – Se os dois bits forem 1 então a sua soma dá sempre origem a um transporte, independentemente de receber um transporte vindo da direita.
- Propagação – Se um dos bits for 1 e o outro for 0 então essa soma dá origem a um transporte se, e só se, receber um transporte vindo do sua direita.

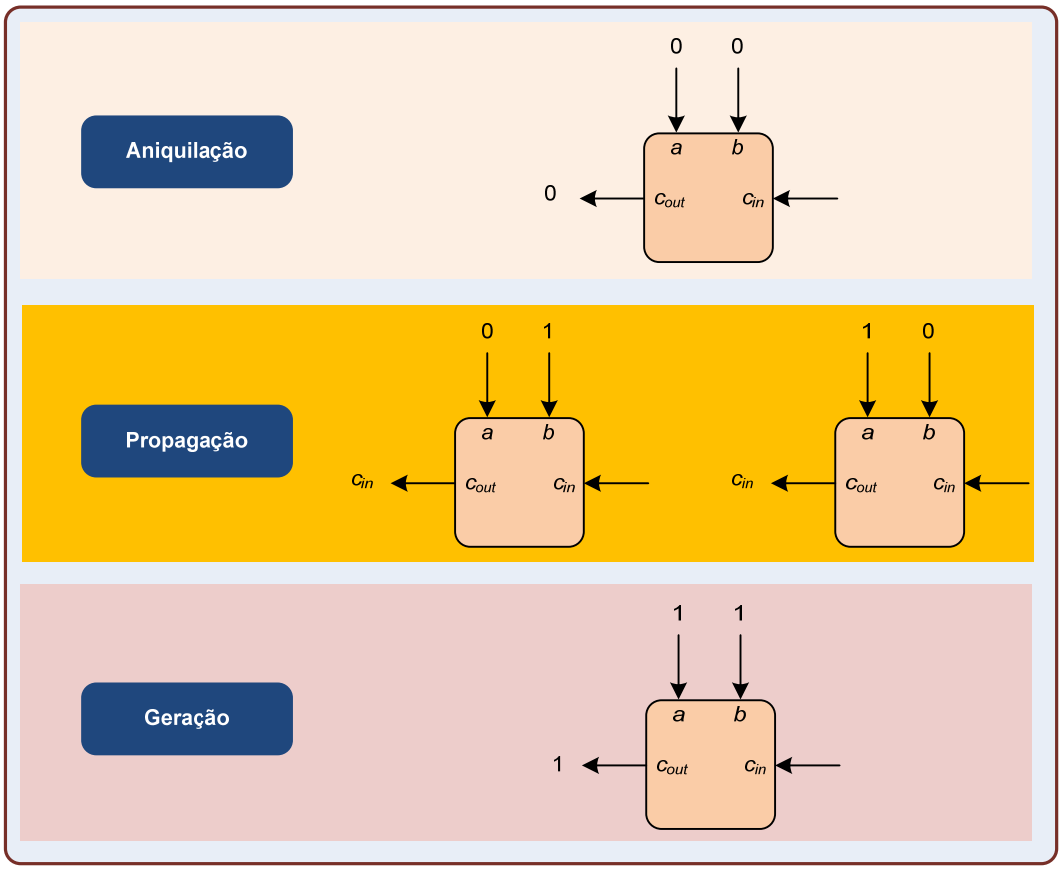


Figura 3.22 – Ilustração do comportamento do bit de transporte (c_{out}) em função dos dois bits a somar (a e b).

Matematicamente os dois últimos casos podem ser expressos pelas seguintes variáveis:

$$\begin{aligned} g_i &= a_i \cdot b_i \\ p_i &= a_i \oplus b_i \end{aligned} . \tag{7}$$

O transporte de um dado bit i pode ser determinado a partir do transporte do bit à direita, c_i, juntamente com o valor de g_i e p_i:

$$c_{i+1} = g_i + p_i \cdot c_i . \tag{8}$$

A soma dos dois bits pode ser obtida usando

$$s_i = p_i \oplus c_i . \tag{9}$$

Tendo em conta (8) observa-se que o XOR presente em (7) pode ser substituído por um OR pois quando os dois bits têm o valor 1 g_i será também 1 o que leva a que c_{i+1} seja 1 independente de p_i. Pode-se então escrever

$$\begin{aligned} g_i &= a_i \cdot b_i \\ p_i &= a_i + b_i \end{aligned} . \tag{10}$$

Os valores de g_i e p_i podem ser calculados em paralelo para todos os bits, sendo o tempo de propagação igual a 1. A área ocupada por estes circuitos de cálculo dos bits de geração e propagação é de 2*N*, para operandos *a* e *b* de *N* bits.

A equação (10) pode-se aplicar recursivamente para obter c_{i+1} apenas a partir de g_k e p_k, para $k \le 1$, e c_0:

$$\begin{aligned} c_{i+1} &= g_i + p_i \cdot c_i \\ &= g_i + p_i \cdot (g_{i-1} + p_{i-1} \cdot c_{i-1}) \\ &= g_i + p_i \cdot g_{i-1} + p_i \cdot p_{i-1} \cdot (g_{i-2} + p_{i-2} \cdot c_{i-2}) \\ &\ldots\ldots \\ &= g_i + p_i \cdot g_{i-1} + p_i \cdot p_{i-1} \cdot g_{i-2} + p_i \cdot p_{i-1} \cdot p_{i-2} \cdot g_{i-3} + \ldots + p_i \cdot p_{i-1} \cdots p_1 \cdot p_0 \cdot c_0 \end{aligned} \quad . \qquad (11)$$

Com base em (11) verifica-se que uma vez calculados os g_i e p_i para cada bit é possível calcular os c_i de todos bits em dois níveis. As portas desses níveis têm no entanto um elevado número de entradas. Como se observa em (11) o cálculo do último bit de transporte consiste numa soma de *N*+1 termos sendo o último desses termos um produto de *N*+1 termos.

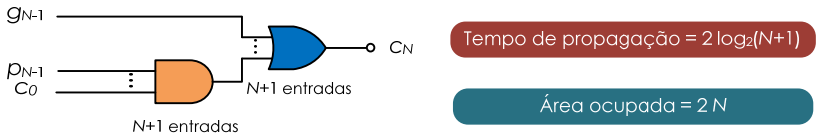


Figura 3.23 – Realização lógica alternativa do somador completo (bloco FA).

De acordo com o modelo de Tyagi, o tempo de propagação do circuito lógico para o cálculo do transporte do bit mais significativo é $2\lceil \log_2(N+1) \rceil + 2$, tendo em conta um tempo de propagação de 2 para o cálculo dos valores de p_i e g_i.

A área ocupada pelo circuito para o cálculo de cada bit de transporte é a soma da área da porta OR com *i*+1 entradas com a das *i* portas AND:

$$\text{área para o cálculo de } c_i = i + \sum_{k=1}^{i} k = i + \frac{(1+i)i}{2} = \frac{3}{2}i + \frac{1}{2}i^2 \quad . \qquad (12)$$

A área necessária para o cálculo de todos os bits de transporte (A_t) é

$$A_t = \sum_{i=1}^{N} \left(\frac{3}{2}i + \frac{1}{2}i^2 \right) = \frac{3}{2}\frac{(1+N)N}{2} + \frac{1}{2}\frac{N(N+1)(2N+1)}{6} = \frac{5}{6}N + N^2 + \frac{1}{6}N^3 \, . \qquad (13)$$

Depois de conhecidos os c_i (e os p_i) basta usar (9) para calcular os s_i, o que demora o tempo de atraso de mais uma porta (um XOR), isto é, 2. As *N* portas XOR correspondentes aos *N* bits do somador ocupam uma área de 2*N*.

O tempo de propagação total é portanto de $5+2\cdot\lceil \log_2(N+1) \rceil$. A área total é

$$\text{área total} = 4,833N + N^2 + 0,166N^3 \quad (14)$$

A Figura 3.24 mostra o circuito lógico usado para implementar o somador Carry-Lookahead, usando como referência o somador Ripple-Carry.

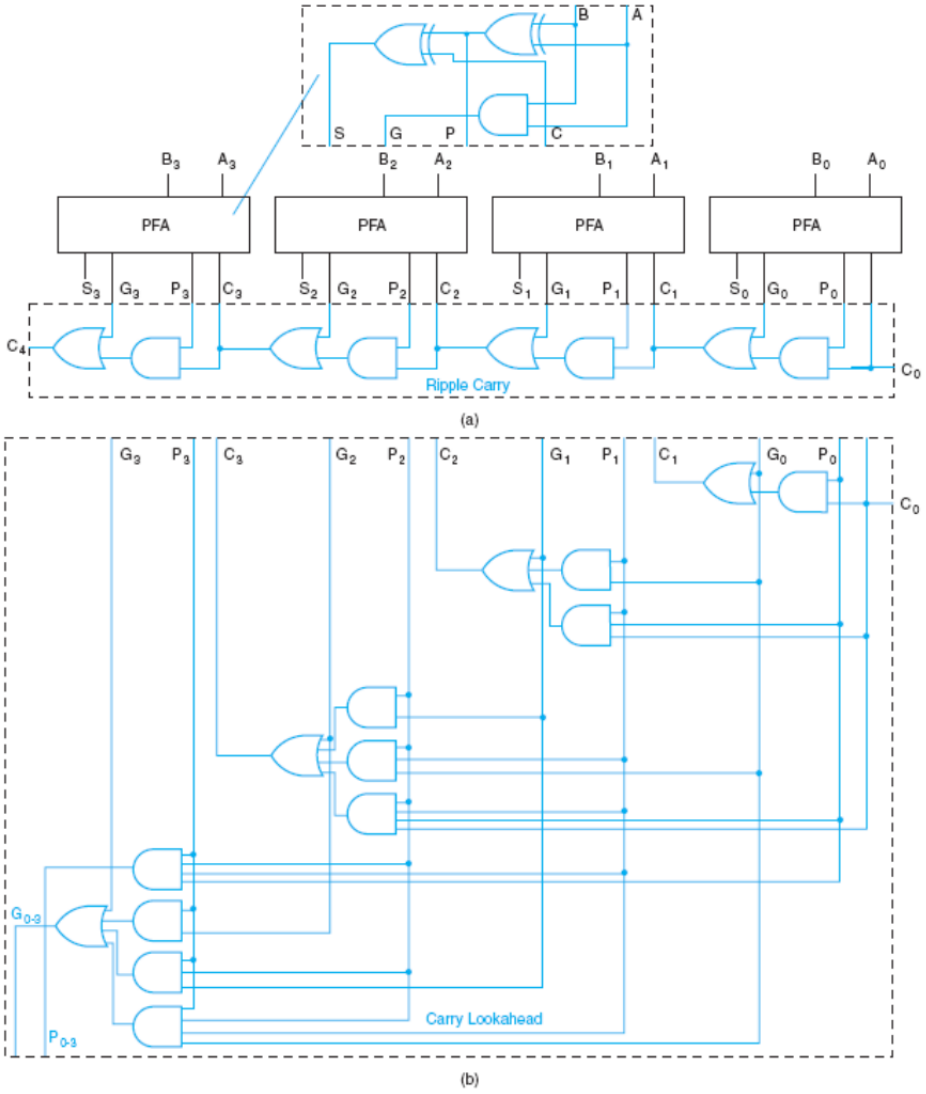


Figura 3.24 – Comparação dos circuitos lógicos para implementação dos somadores Carry-Lookahead e Ripple-Carry.

Este somador tem a vantagem de ser mais rápido do que os somadores Ripple-Carry e Carry-Select mas tem a desvantagem de possuir portas lógicas com muitas entradas e portanto ocupar uma grande área. Outra desvantagem está relacionada com o grande *fan-in* (N+1 portas ligadas à entrada de uma mesma porta) e *fan-out* (a saída de uma porta ligada a N+1 entradas de outras portas) requerido das portas lógicas.

3.2.4 Somador Carry-Lookahead em Bloco

Uma solução para o problema referido na secção anterior passa pela combinação das técnicas Ripple-Carry e Carry-Lookahead para o projecto dos circuitos de soma. Os bits são agrupados em blocos. O cálculo em cada bloco é efectuado por um somador Carry-Lookahead sendo o bit de transporte propagado entre os blocos que assim constituem um somador Ripple-Carry (Figura 3.25).

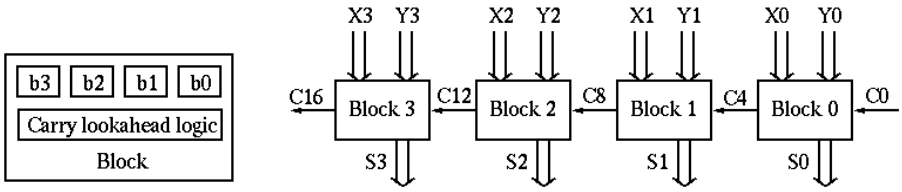


Figura 3.25 – Estrutura de um somador transporte Lookahed em bloco.

É possível também usar dois níveis de somadores Carry-Lookahead como ilustrado na Figura 3.26.

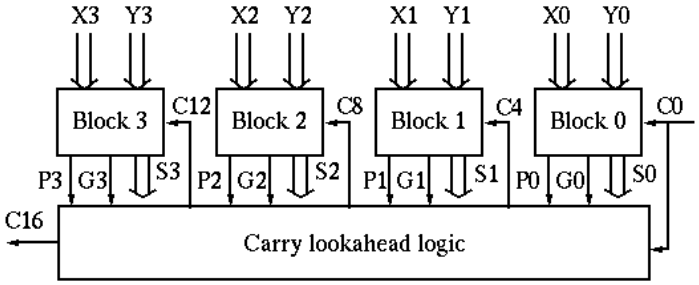


Figura 3.26 – Somador Carry-Lookahead de dois níveis.

3.2.5 Somador Carry-Lookahead em Árvore

Uma outra solução, frequentemente utilizada, usa esquemas hierárquicos para o projecto de somadores Carry-Lookahead em árvore. A Figura 3.27 mostra o primeiro passo na construção deste circuito – o cálculo em paralelo dos bits de geração e propagação correspondentes a cada par de bits dos operandos.

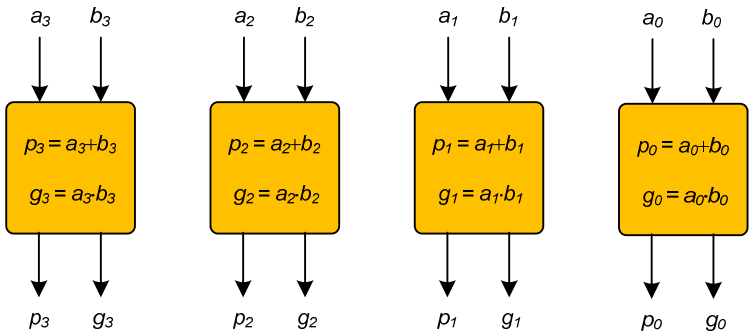


Figura 3.27 – Diagrama de blocos do circuito somador Carry-Lookahead. Passo 1 da exemplificação do funcionamento.

Esses bits são então usados para calcular os bits de geração e propagação de grupos de dois bits consecutivos, no caso de árvores binárias. No caso da Figura 3.29 são formados dois grupos: o da direita, constituído pelos bits 0 e 1; o da esquerda constituído pelos bits 2 e 3. Para cada um desses grupos é possível determinar os bits de geração e propagação que determinarão o bit de transporte de saída do grupo de bits respectivo, dado o transporte de entrada desse grupo (Figura 3.28).

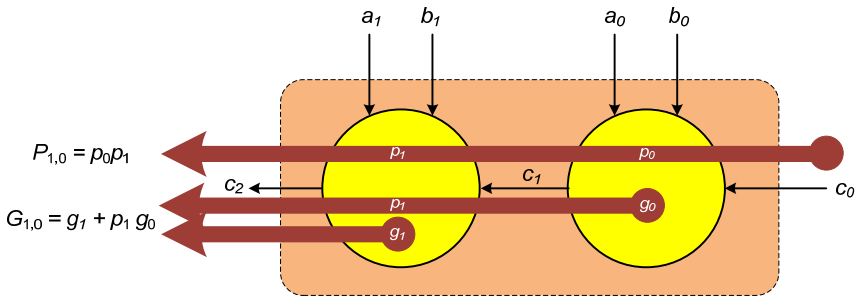


Figura 3.28 – Ilustração do cálculo dos bits de geração e propagação de um grupo de dois bits.

Neste caso tem-se, para determinação dos bits de geração e propagação de grupo a partir dos bits de geração e propagação iniciais, as seguintes expressões:

$$\begin{aligned} P_{1,0} &= p_0 \cdot p_1 \\ G_{1,0} &= g_1 + p_1 \cdot g_0 \\ &\cdots \\ P_{3,2} &= p_2 \cdot p_3 \\ G_{3,2} &= g_3 + p_3 \cdot g_2 \end{aligned} \qquad (15)$$

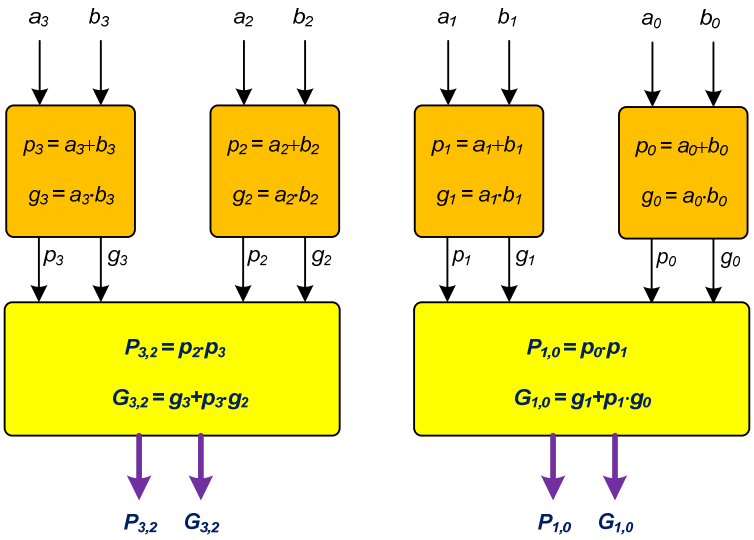


Figura 3.29 – Diagrama de blocos do circuito somador Carry-Lookahead. Passo 2 da exemplificação do funcionamento.

De forma semelhante, com base nesses dois grupos, é possível formar um outro grupo que os engloba e que representa todos os bits todas das palavras a somar (Figura 3.30). Os bits de geração e propagação deste grupo maior podem ser calculados de forma semelhante, a partir dos bits de geração e propagação dos dois grupos elementares:

$$\begin{aligned} P_{3,0} &= P_{1,0} \cdot P_{3,2} \\ G_{3,0} &= G_{3,0} + P_{3,2} \cdot G_{1,0} \end{aligned} \quad . \tag{16}$$

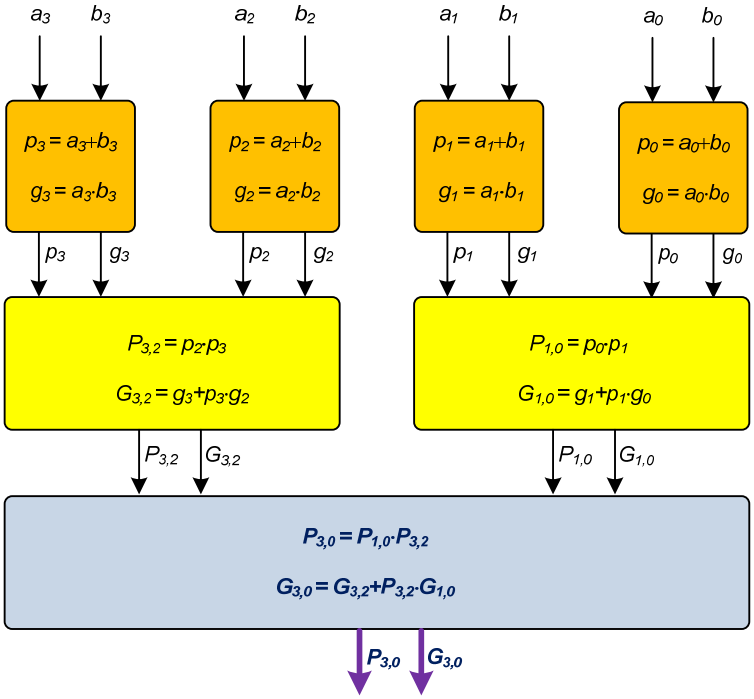


Figura 3.30 – Diagrama de blocos do circuito somador Carry-Lookahead. Passo 3 da exemplificação do funcionamento.

Generalizando, é possível calcular os bits de geração e propagação de um grupo de bits a partir dos sub-grupos de nível inferior calculando:

$$\begin{aligned} P_{k,i} &= P_{k,j+1} \cdot P_{j,i} \\ G_{k,i} &= G_{k,j+1} + P_{k,j+1} \cdot G_{j,i} \end{aligned} \quad , \tag{17}$$

em que $i < j < j+1 < k$.

Tendo calculado os bits de geração e propagação é possível calcular os bits de transporte. Isso é feito começando no grupo de nível mais alto até aos grupos de nível mais baixo. A Figura 3.31, por exemplo, mostra o cálculo dos bits c_3 e c_5 pelo grupo de mais alto nível.

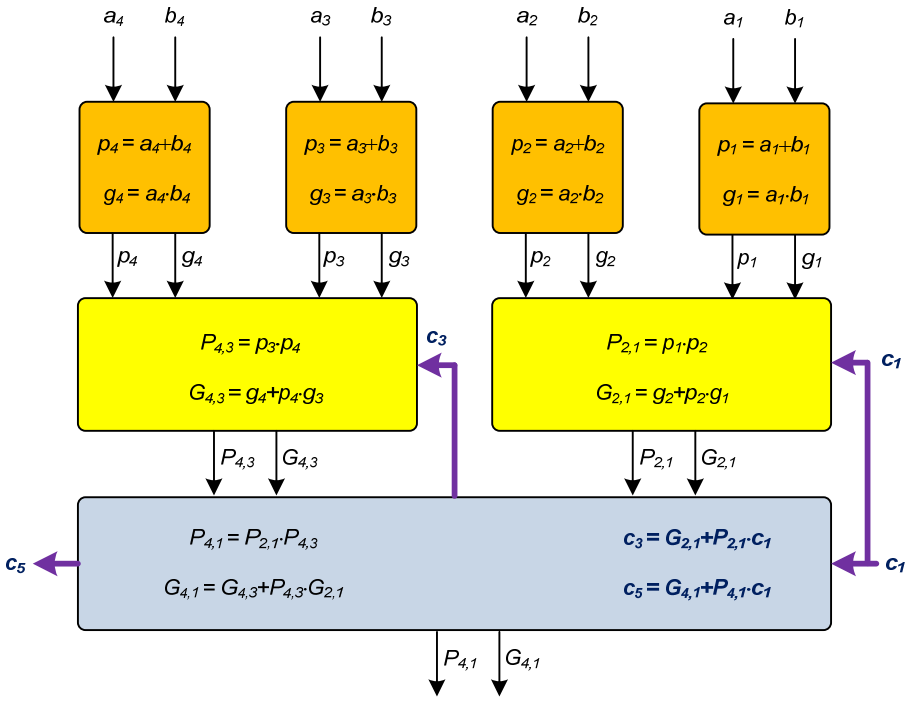


Figura 3.31 – Diagrama de blocos do circuito somador Carry-Lookahead. Passo 4 da exemplificação do funcionamento.

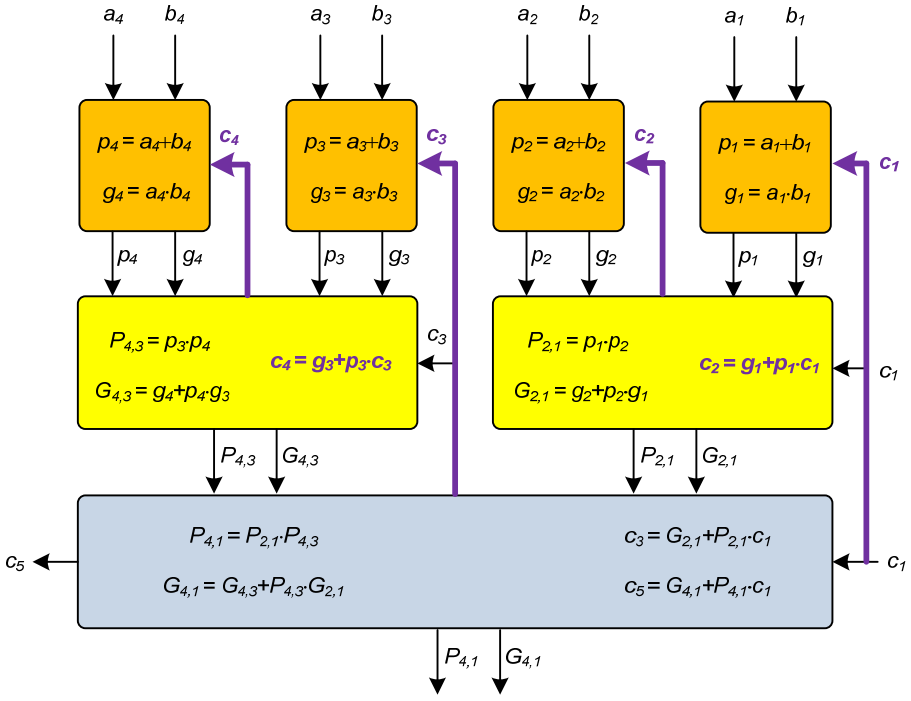


Figura 3.32 – Diagrama de blocos do circuito somador Carry-Lookahead. Passo 5 da exemplificação do funcionamento.

O cálculo dos bits de transporte é efectuado, de forma geral:

$$c_{k+1} = G_{k,i} + P_{k,i} \cdot c_i \,. \tag{18}$$

A Figura 3.32 mostra o cálculo dos bits c_2 e c_4 realizado pelos dois grupos de 1º nível.

Finalmente, tendo todos os bits de transporte calculados, é possível determinar os bits de soma usando (2) conforme se ilustra na Figura 3.33.

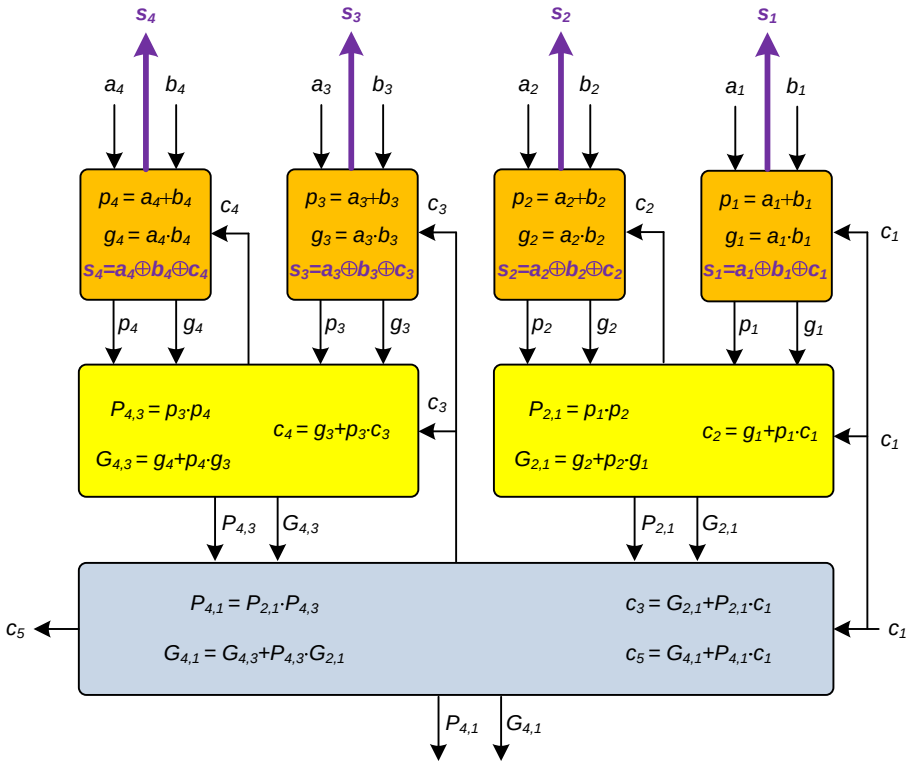


Figura 3.33 – Diagrama de blocos do circuito somador Carry-Lookahead. Passo 6 da exemplificação do funcionamento.

Em relação ao tempo de propagação deste circuito somador Carry-Lookahead em árvore, o caminho crítico representado na Figura 3.34 correspondente à determinação do bit de soma s_4.

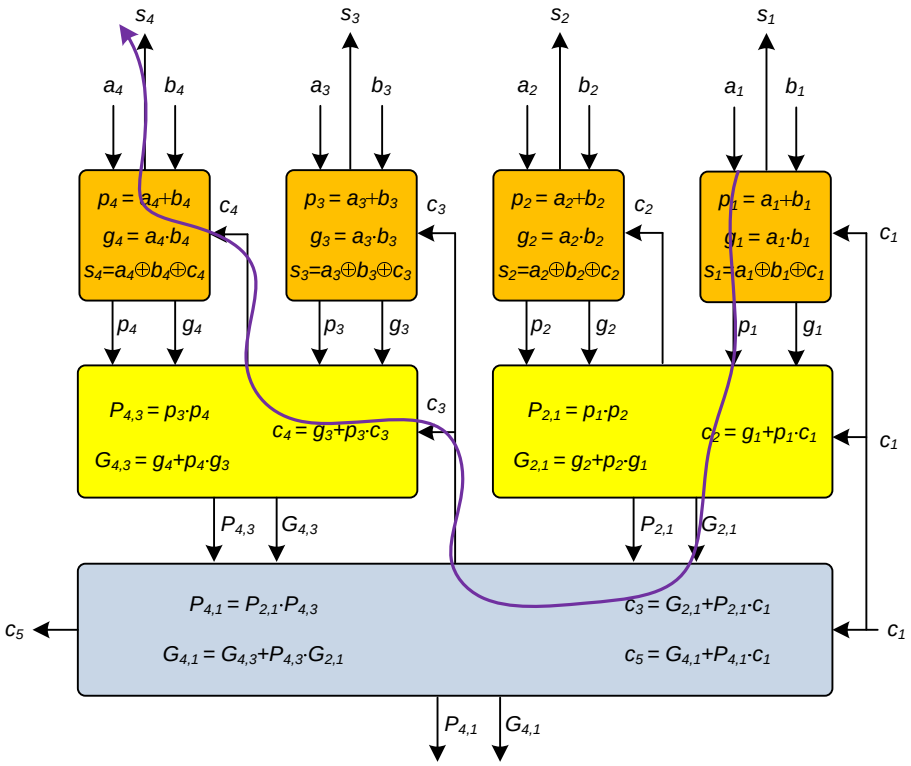


Figura 3.34 – Diagrama de blocos do circuito somador Carry-Lookahead. Caminho crítico para o cálculo da soma.

Para determinação do tempo de propagação do circuito de soma do tipo Carry-Lookahead há que considerar o seguinte:

- O cálculo de *p* e *g* demora 1 medida de tempo independentemente do número de bits, pois esse cálculo é feito em paralelo.
- Em cada nível da árvore os *P* e *G* são calculados a partir dos *P* e *G* (ou *p* e *g*) que vêm do nível anterior. Esse cálculo tem um atraso de 2 e é feito em paralelo para todos os grupos do mesmo nível.
    - Exemplo: $G_{2,1} = g_2 + p_2 \times g_1$.
- Cada árvore binária tem $\log_2(N)$ níveis em que *N* é o número de bits, mas para o tempo de propagação dos bits de soma de nível inferior da árvore não efectua processamento necessário, pelo que esse nível não deve ser contabilizado.
- Depois de chegar ao nível inferior da árvore tendo sido calculados todos os *P*s e *G*s, é preciso percorrer a árvore no sentido inverso para calcular os bits de transporte (e no andar final o bit da soma). Cada um desses passos tem um atraso de 2.

O tempo total é portanto = $2\times2\times\log_2(N) -2+1+2 = 4\times\log_2(N) -1+2$.

O tempo de propagação do bit de transporte (T_t) que saí do somador é dado por

$$T_t = 2\cdot\log(N)+1, \qquad (19)$$

pois neste caso basta percorrer todos os níveis da árvore no sentido descendente.

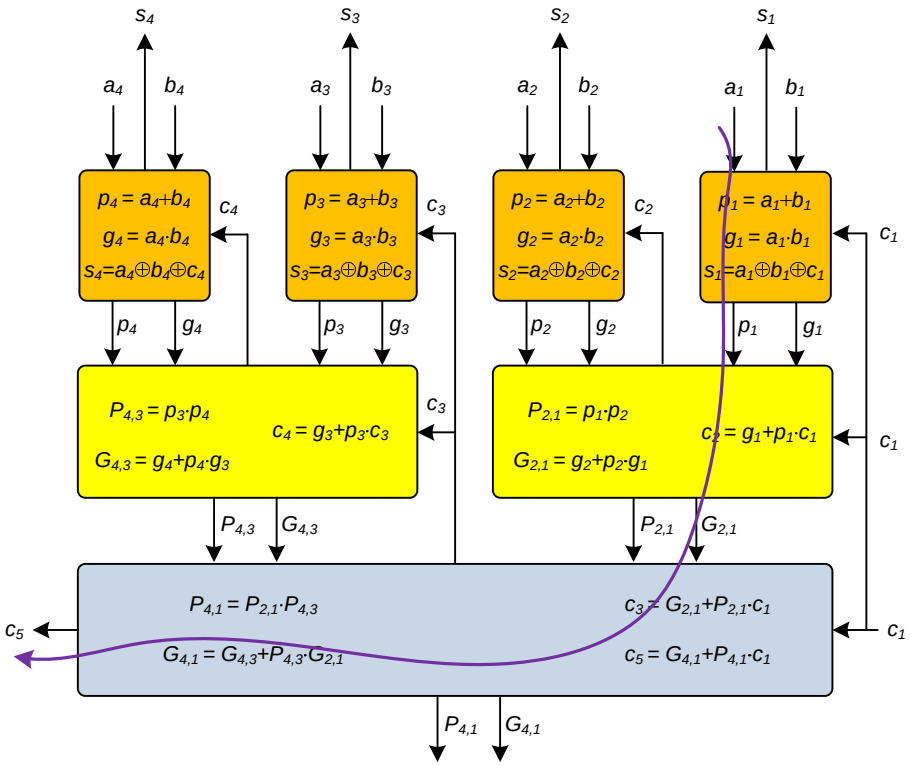


Figura 3.35 – Diagrama de blocos do circuito somador Carry-Lookahead. Percurso crítico para o cálculo do bit de transporte de saída.

3.2.6 Somador Carry-Save

Os somadores vistos até agora permitem a soma de dois números de cada vez. Para se somar mais do que dois números usando esses somadores há que agrupá-los dois a dois até todos terem sido somados. Há, no entanto, circuitos somadores que são vocacionados para efectuar a soma de três ou mais números apresentando nessas situações vantagens em termos de rapidez e tamanho. O que veremos nesta secção é chamado de somador Carry-Save.

Um somador Carry-Save, em vez de usar o bit de transporte que sai da soma de um bit para a soma do bit seguinte, guarda (*save*) esse bit para ser usado mais tarde. Cada somador de bit completo efectua a soma do que no fundo são 3 bits, produzindo duas saídas – o bit de soma e o bit de transporte.

A Figura 3.36 ilustra essa diferença. Na configuração da esquerda, a saída de transporte de cada somador de bit completo está ligada à entrada de transporte do somador de bit completo colocado à sua direita. Esta organização constitui um somador Ripple-Carry, como foi visto anteriormente. O conjunto de saídas de soma dos somadores completos constitui o resultado dos dois números de entrada.

A diferença entre este diagrama de blocos e o diagrama apresentado na Figura 3.11 é que neste caso é considerada a existência de uma entrada de transporte (c_0) que pode ser conectada à saída

de transporte de um outro somador caso façam parte de um somador com um maior número de bits. Caso não haja essa conexão basta fazer $c_0 = 0$ para que o funcionamento nos dois casos seja idêntico.

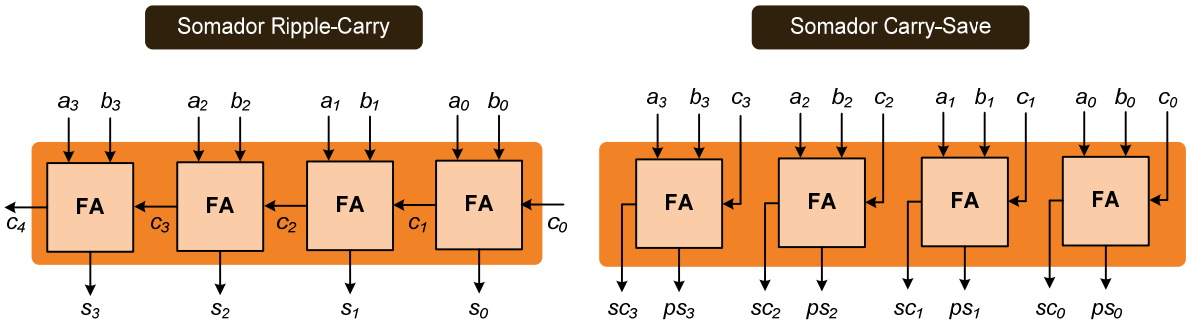


Figura 3.36 – Diagrama de blocos do circuito somador do tipo Ripple-Carry e da unidade Carry-Save.

Na configuração da direita da Figura 3.36 cada somador de bit completo tem uma entrada de transporte que vem do "exterior" e a sua saída de transporte não está conectada a nenhum dos outros somadores. Essa saída, designada por *sc* (*shift-carry*), vai para o "exterior". As quatro saídas da soma dos somadores de bit completos vão também para o exterior só que neste caso não constituem, só por si o resultado da soma. Por essa razão são designadas por *ps* (*partial-sum*). Só o conjunto das 4 saídas *ps* e das 4 saídas *sc* constitui o resultado da soma. Esse resultado não está no entanto representado da forma tradicional (binária), usando 4 bits, como no caso da saída do somador Ripple-Carry.

Pode-se encarar a saída de cada somador de bit completo como sendo a soma do número de bits igual a 1 presente nas 3 entradas do somador completo como ilustrado na Tabela 3.6. Essa soma, que pode assumir um valor entre 0 e 3, pode ser representada por dois bits (*ps* e *sc*).

Tabela 3.6 – Relação entre a entrada e a saída de um bloco FA do ponto de vista de um somador Carry-Save.

Entradas			Número de 1s	Saídas	
a_i	b_i	c_i		sc_i	ps_i
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

Na unidade Carry-Save todos os somadores completos podem operar em paralelo. Para se obter o valor da soma utilizando representação binária é preciso que eventualmente os bits de transporte se propaguem da direita para a esquerda até ao último bit. Isso pode ser feito, por exemplo, com um

somador Ripple-Carry que some os bits *ps* com os bits *sc* deslocados para a esquerda de uma unidade como ilustrado na Figura 3.37.

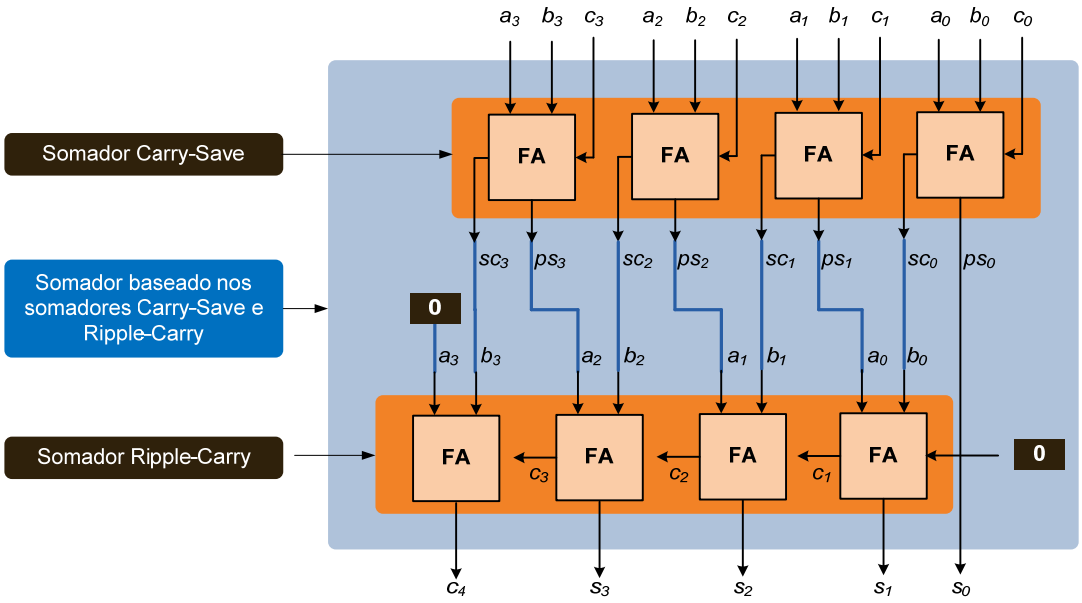


Figura 3.37 – Somador Carry-Save.

O somador Carry-Save de 3 números apresentado na Figura 3.37 não é mais rápido nem menor do que um somador formado por dois somadores Ripple-Carry. A vantagem de usar um somador Carry-Save surge quando se tem uma maior quantidade de números a somar pois só será necessário, em qualquer caso, o uso de um somador Ripple-Carry para fazer a propagação do bit de transporte da direita para a esquerda.

Para ilustrar a constituição e funcionamento de somadores Carry-Save capazes de somar mais números torna-se mais simples utilizar uma notação um pouco diferente da usada até aqui. Usam-se pequenos círculos para representar os bits e a sua localização para indicar quais são somados com quais. Considere, por exemplo, a Figura 3.38. Aí representam-se os dois números a somar, com 6 bits cada, por duas linhas horizontais paralelas constituídas por 6 círculos. Numa terceira linha pode-se representar o bit de transporte de entrada do somador. No caso de ser utilizado um somador Ripple-Carry, por exemplo, a saída são 7 bits representados por 7 círculos abaixo da linha a tracejado. Essa ilustração traduz, no fundo, a disposição dos bits numa soma efectuada com papel e lápis.

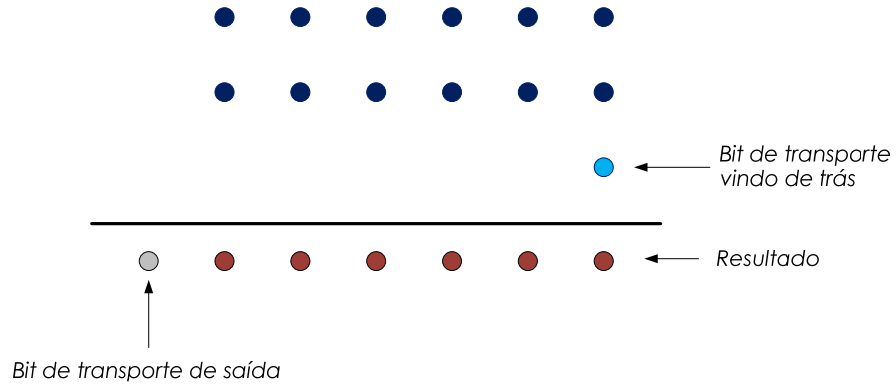


Figura 3.38 – Representação, usando pontos, para representar o ponto de vista da soma de dois números efectuada por um somador Ripple-Carry.

A Figura 3.39 ilustra o caso da soma de 3 números de 6 bits cada usando um somador Carry-Save. As 3 primeiras linhas representam os 3 números a somar. Os 6 rectângulos verticais são 6 somadores completos encarados como somadores Carry-Save de 1 bit. São blocos que recebem 3 bits e produzem 2 bits, como ilustrado no esquema da direita da Figura 3.36.

No seu conjunto os 6 somadores de 1 bit produzem 12 bits de saída – 6 bits de soma parcial (*ps*), representados numa linha e 6 bits de transporte (*sc*) representados numa linha abaixo e deslocados de uma posição para a esquerda.

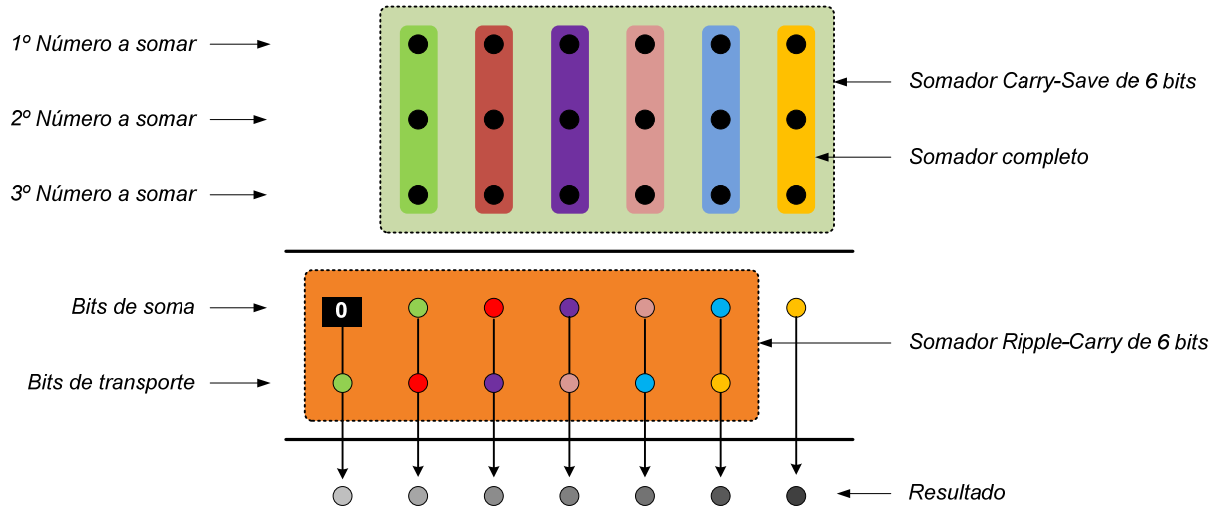


Figura 3.39 – Ilustração do funcionamento de um somador constituído por um domador Carry-Save e um somador Ripple-Carry.

O exemplo apresentado ilustra como um somador Carry-Save é capaz de reduzir 3 números a dois números. Esses dois números são depois adicionados com um somador em que haja propagação do bit de transporte (Ripple-Carry, Select-Carry, Carry-Lookahead, etc.).

Quando se pretende construir um somador para somar mais do que 3 números é possível usar vários somadores Carry-Save, organizados em diferentes níveis hierárquicos, para ir reduzindo o número de

linhas até se ter duas. Existem diferentes soluções possíveis e que consistem em agrupar os números a somar em conjuntos de 3 e usar somadores Carry-Save para reduzir cada conjunto de 3 num conjunto de 2. De seguida volta a agrupar-se novamente os números iniciais e os resultados das primeiras somas em novos grupos de 3 de forma a serem reduzidos a grupos de 2 através do uso de somadores Carry-Save. A Figura 3.40 ilustra a soma de 7 números de 6 bits cada.

Mais informação sobre os somadores, nomeadamente sobre o somador Carry-Save, pode ser encontrada em [6].

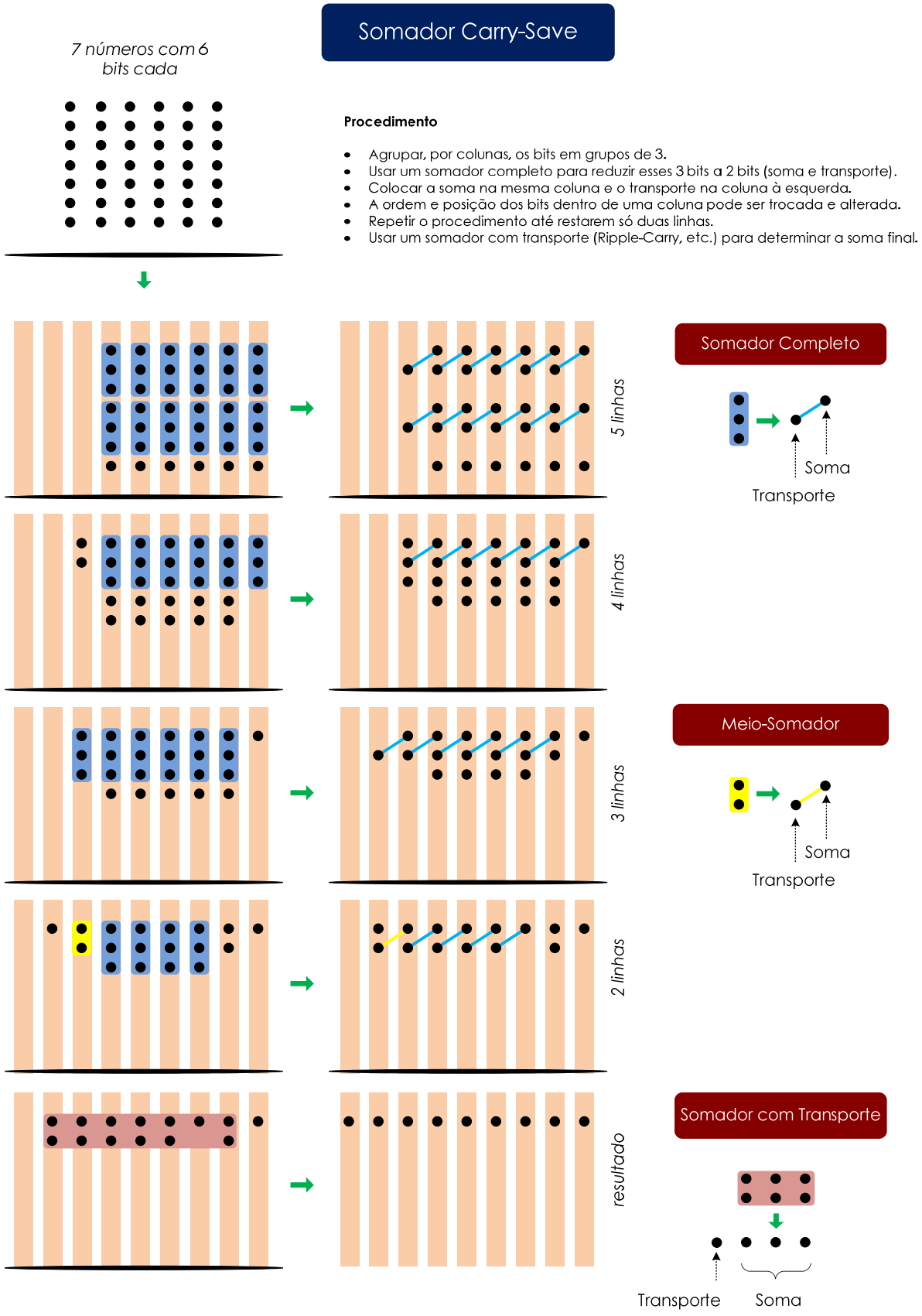


Figura 3.40 – Ilustração de um somador Carry-Save que soma 7 números de 6 bits cada.

3.3 Subtracção

A subtracção pode ser feita por circuitos especializados que funcionam de uma forma semelhante aos somadores ou pode ser feita por um circuito somador e um circuito que calcula o complemento para dois. Esta técnica consiste em realizar a subtracção de dois números através da soma de um número com o simétrico do outro.

3.3.1 Subtractor Ripple-Carry

A subtracção de dois números pode ser realizada com topologias semelhantes às utilizadas nos circuitos somadores apresentados. Em vez de somadores completos que somam duas palavras de um bit tendo em conta o transporte que vem de trás (à direita) e produzindo o um transporte para o bit seguinte (à esquerda) é substituido por "subtractores completos" que subtraem dois bits.

Cada vez que se tem que subtrair 1 de 0 (0 - 1) tem de se pedir emprestado ao bit da direita um "1". O resultado dessa subtração é portanto 1 e "vai 1" para ser subtraído do bit à direita. Nos outros casos não há lugar a empréstimo: 1 - 1 = 0, 1 - 0 = 1, 0 - 0 = 0. A Figura 3.41 ilustra o caso da subtracção 10 - 3.

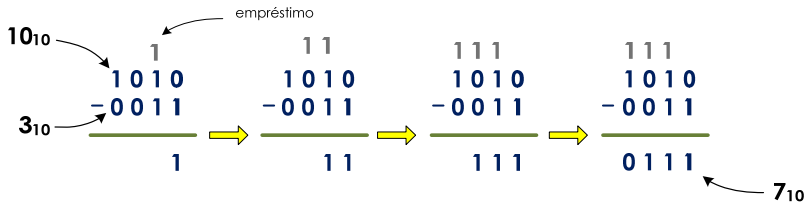


Figura 3.41 – Ilustração da subtracção de dois números em base 2.

A Figura 3.42 mostra o diagrama de blocos de um circuito para realizar este procedimentos. São usados neste caso "subtractores completos" (FS, do inglês *full subtractor*). Cado bloco, além dos dois bits a subtrair tem os bits de empréstimo de entrada e de saída (e_i).

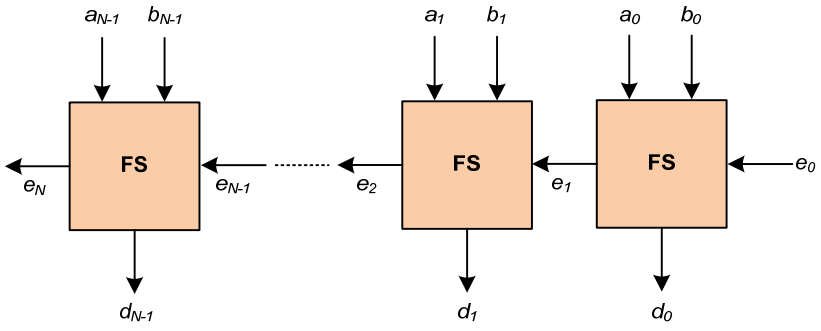


Figura 3.42 – Diagrama de blocos do circuito subtractror do tipo "Ripple-Carry".

A tabela de verdade do subtractor-completo é a apresentada na Tabela 3.7.

Tabela 3.7 – Tabela de Verdade do subtractor completo.

Entradas			Saídas	
a_i	b_i	e_i	e_{i+1}	d_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

A saída d_i do subtractor completo pode ser obtida usando

$$d_i = a_i \oplus b_i \oplus e_i \tag{19}$$

A saída e_{i+1} pode ser obtida com um mapa de Karnaugh como o representado na Figura 3.13.

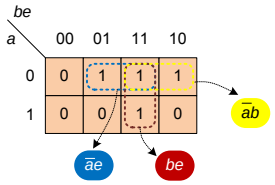


Figura 3.43 – Mapas de Karnaugh para o problema proposto.

A expressão matemática é portanto

$$e_{i+1} = \overline{a_i} \cdot b_i + \overline{a_i} \cdot e_i + b_i \cdot e_i \,. \tag{19}$$

Esta expressão pode ser escrita de outra forma colocando a variável e_i em evidência:

$$e_{i+1} = \overline{a_i} \cdot b_i + e_i \cdot \left(\overline{a_i} + b_i\right). \tag{19}$$

Esta função lógica pode ser implementada por um conjunto de portas lógicas interligadas conforme se ilustra na Figura 3.44.

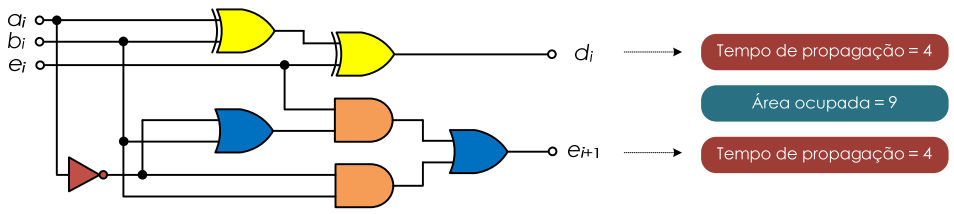


Figura 3.44 – Realização lógica do subtractor completo (bloco FS).

O circuito que calcula o bit de diferença (d_i) é constituído por duas portas XOR o que dá uma área ocupada de 4 e um tempo de propagação de 4. Já o circuito que efectua o cálculo do bit de

empréstimo (e_{i+1}) é constituído por uma porta NOT, duas portas OR e duas portas AND que operam em paralelo. Assim a área ocupada é de 5 e o tempo de propagação é de 4. No total a área ocupada é de 9 (4+5).

3.3.2 Subtractor usando um somador e o complemento para 2

Qualquer dos circuitos somadores vistos pode ser combinado com um circuito que calcula o complemento para 2 do segundo operando para se obter a diferença. Esta é a técnica que é mais utilizada pois leva a uma menor área total de implementação da ALU com um tempo de propagação ligeiramente superior. Os circuitos somadores/subtractores têm uma entrada adicional que determina a operação a realizar.

A Figura 3.45 ilustra este circuito baseado num somador ripple-Carry no caso de palavras de 8 bits. O complemento para dois consiste em inverter os 0s e 1s o que é feito com portas ou-exclusivo ligadas ao sinal de controlo da operação a realizar (S) e aos bits do segundo operando. Para além disso é preciso somar 1 o que é feito através do bit de transporte inicial (mais à direita) ligando-se aí directamente o sinal de controlo da operação.

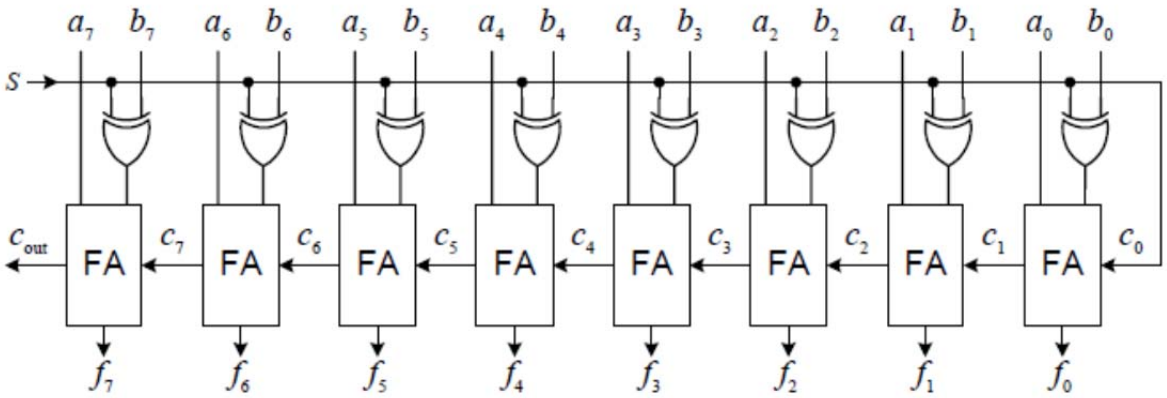


Figura 3.45 – Somador /Subtractor Ripple Carry.

3.4 Multiplicação

A multiplicação é uma das operações aritméticas mais comuns. A sua realização baseia-se, em grande parte, na adição. A Figura 3.46 relembra como se multiplicam dois números em base 2 usando papel e lápis.

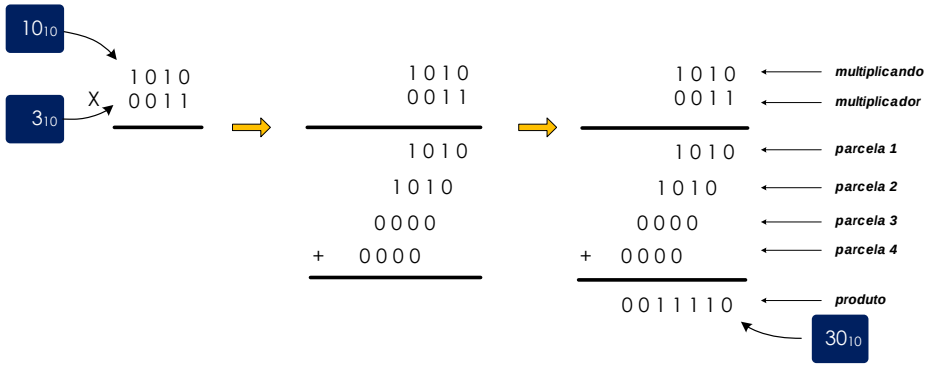


Figura 3.46 – Ilustração da multiplicação de dois números em base 2.

O multiplicando é multiplicado por cada um dos bits do multiplicador dando origem a um número de termos igual ao número de bits do multiplicador. Esses termos são deslocados sucessivamente para a esquerda de uma posição pois correspondem a bits do multiplicador com pesos cada vez maiores. Os termos assim obtidos e posicionados são somados coluna a coluna a começar da direita tendo em conta o transporte que pode vir da coluna anterior.

A Figura 3.47 apresenta a designação comum dada aos diferentes bits do multiplicando, multiplicador e produto. Apresenta-se também a constituição dos diferentes produtos parciais de dois bits.

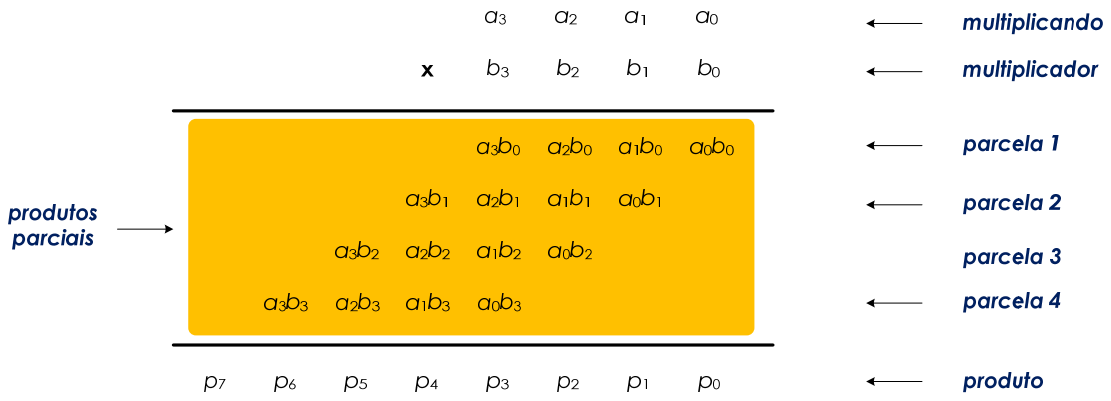


Figura 3.47 – Designação dos termos usados na multiplicação.

A multiplicação de números negativos usando papel e lápis é efectuada encarando o sinal e o número de forma separada. Os números, sem bit de sinal, são multiplicados como descrito e o sinal do produto é determinado, de forma independente a partir dos bits de sinal do multiplicando e do multiplicador. Num processador, os números negativos são em geral representados usando o complemento para dois e não acrescentando um bit de sinal à esquerda do número. A multiplicação usando complemento para dois não pode ser feita com o mesmo algoritmo que usamos para a multiplicação de números positivos.

Na criação de um circuito electrónico digital para a realização da multiplicação existem dois caminhos. O primeiro caminho consiste em converter os números a multiplicar da representação de complemento para 2 para uma representação com bit de sinal separado, efectuada a multiplicação

do valor absoluto dos números e o cálculo do complemento para dois do resultado caso esse seja negativo (tendo em conta os bits de sinal). O outro caminho é usar um circuito capaz de multiplicar os números directamente a partir da sua representação em complemento para dois mas utilizando um algoritmo diferente do usado para a multiplicação com papel e lápis.

Tal como na adição, existem diversos circuitos para a realização da multiplicação de números representados de forma binária. De seguida apresentam-se alguns desses circuitos. Eles apresentam compromissos em termos de propagação, área ocupada e complexidade das interligações entre os módulos.

3.4.1 Arquitecturas Matriciais

A Figura 3.48 mostra o diagrama de blocos de um multiplicador matricial que calcula a soma entre dois termos usando um somador Ripple-Carry constituído por somadores completos (FA) e meio-somadores (HA). Os produtos parciais constituídos pelo produto de um bit do multiplicando por um bit do multiplicador são calculados usando uma porta lógica AND. A segunda soma (entre o resultado da primeira e o terceiro termo) é válida assim que tiver sido completada a soma do segundo bit no somador Ripple-Carry anterior.

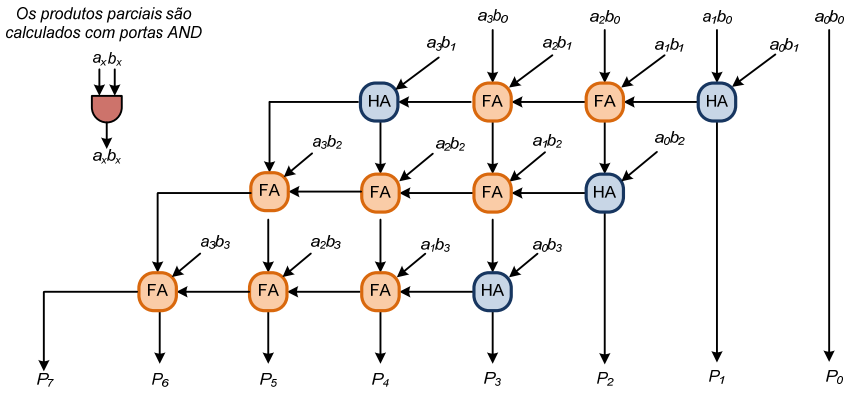


Figura 3.48 – Diagrama de blocos de um multiplicador matricial do tipo Ripple-Carry.

Este circuito é bastante lento devido à propagação do bit de transporte. Uma alternativa consiste em usar somadores Carry-Save em vez de somadores Ripple-Carry.

Os somadores Carry-Save podem ser usados para realizar um multiplicador matricial da forma ilustrada na Figura 3.49.

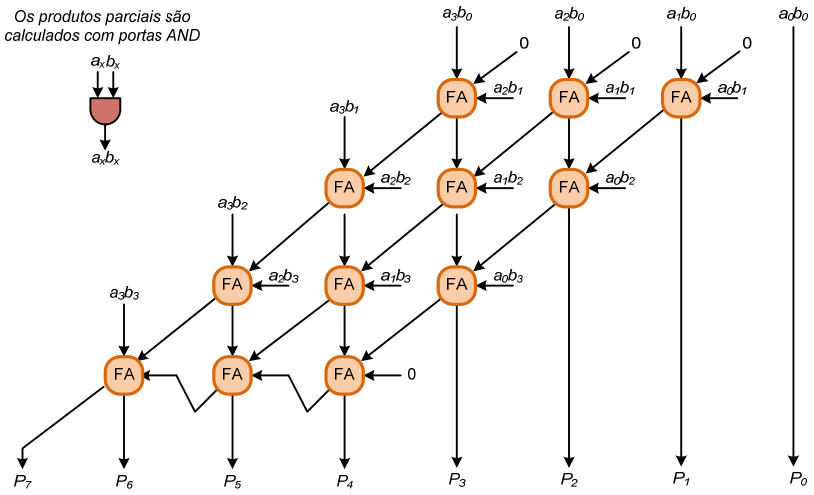


Figura 3.49 – Multiplicador matricial usando a filosofia do somador Carry-Save.

3.4.2 Arquitecturas em Árvore

Outra forma de realizar um multiplicador consiste em utilizar uma arquitectura em árvore. Uma dessas arquitecturas é chamada de arquitectura em árvore de Wallace e consiste em reorganizar os produtos parciais da multiplicação de forma a formarem uma pirâmide invertida (uma árvore) como ilustra a Figura 3.50. A reorganização dos produtos parciais consiste em deslocar verticalmente os produtos parciais de modo a estarem o mais em cima possível.

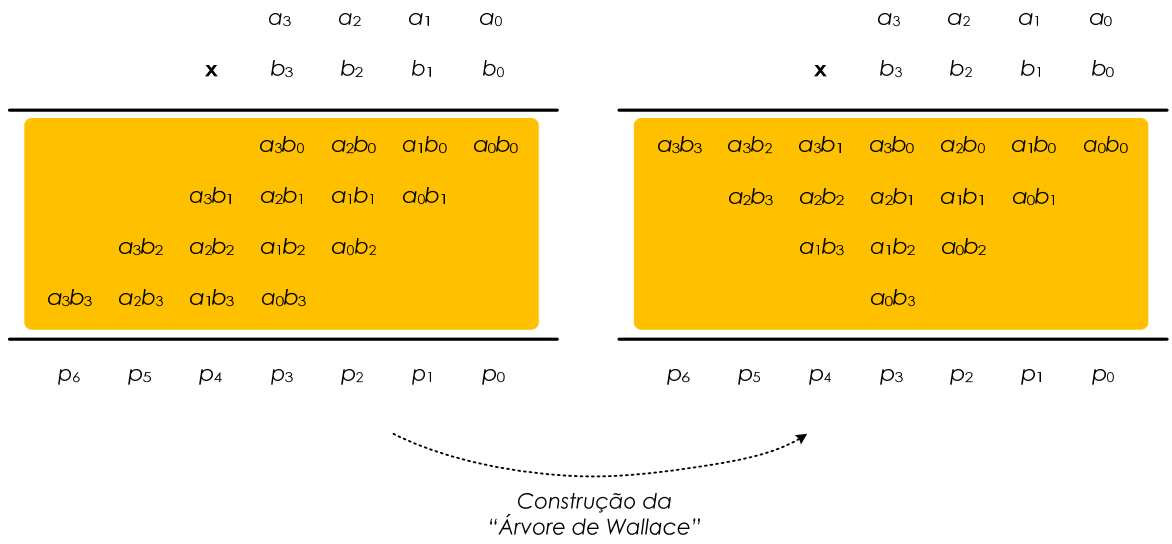


Figura 3.50 – Ilustração do processo de formação da árvore de Wallace no multiplicador com o mesmo nome.

O procedimento subsequente para construção do multiplicador consiste em usar somadores completos (soma de 3 bits: 2 bits mais o transporte de entrada) ou meio-somadores (soma de 2 bits), para ir reduzindo as linhas de produtos parciais a começar pela linha inferior. Deve-se tentar usar de cada vez o menor número de somadores e se possível meio-somadores em vez de somadores completos de forma a eliminar uma linha de cada vez. Há também que ter em atenção que a soma

de 2 ou 3 bits dá origem a um transporte que deve ser colocado na coluna imediatamente à esquerda, por baixo dos produtos parciais já existentes nessa coluna.

Assim sendo o 1º passo é usar um meio-somador para somar os produtos parciais a_1b_2 e a_0b_3. Como se observa na Figura 3.51, a inclusão deste meio-somador reduz o número de produtos parciais na coluna do meio (de 4 para 3) mas, devido ao bit de transporte, aumenta o número de bits na coluna imediatamente à esquerda (de 3 para 4). Não é assim cumprido o objectivo de eliminar a última linha de produtos parciais.

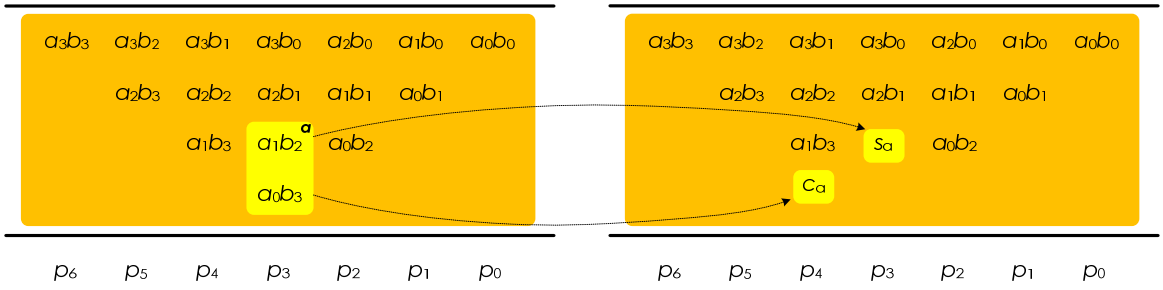


Figura 3.51 – Ilustração da inclusão de um meio-somador para efectuar a soma dos termos a_1b_2 e a_0b_3 num multiplicador de Wallace.

Para que isso aconteça é necessário usar um segundo meio somador para adicionar os produtos parciais a_2b_2 e a_1b_3 como ilustrado na Figura 3.52.

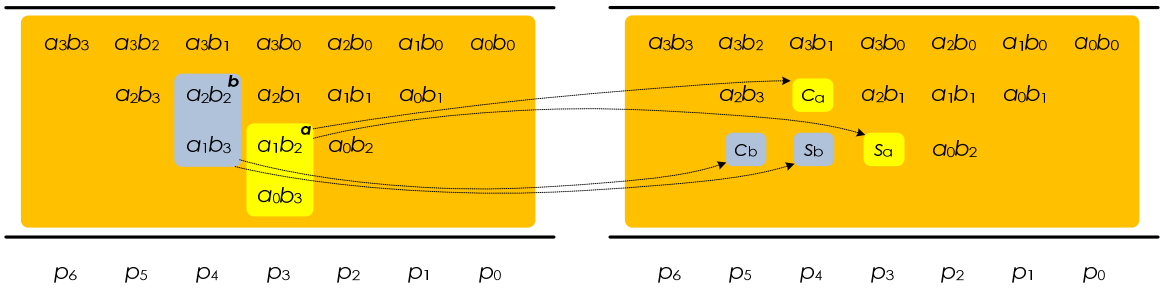


Figura 3.52 – Ilustração do 1º passo na construção do multiplicador de Wallace (eliminação da 4ª linha de produtos parciais).

Como se observa na metade direita da Figura 3.52, existem agora só 3 linhas de produtos parciais. O passo seguinte será eliminar de novo a última dessas linhas. Isso pode ser feito com 1 meio-somador e 3 somadores completos (Figura 3.53).

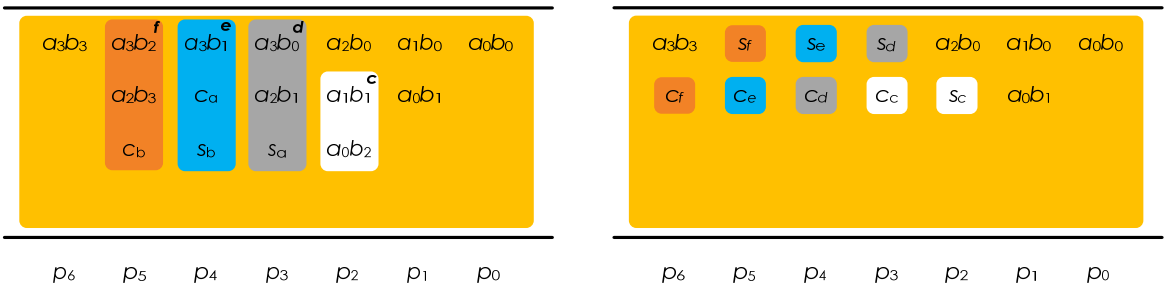


Figura 3.53 – Ilustração do 2º passo na construção do multiplicador de Wallace (eliminação da 3ª linha de produtos parciais).

Finalmente pode-se usar somador Ripple-Carry para efectuar soma das duas linhas restantes (Figura 3.54).

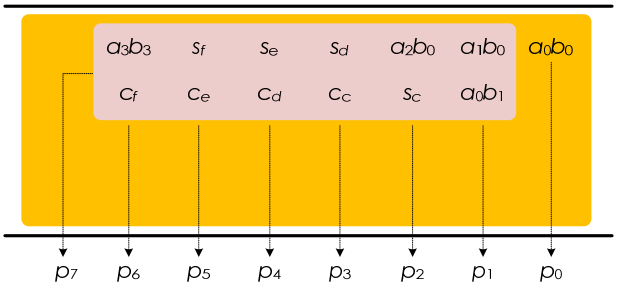


Figura 3.54 – Ilustração do 3º passo na construção do multiplicador de Wallace (soma das duas últimas linhas que restam).

É possível agora criar o multiplicador de Wallace com base nos somadores determinados (Figura 3.55).

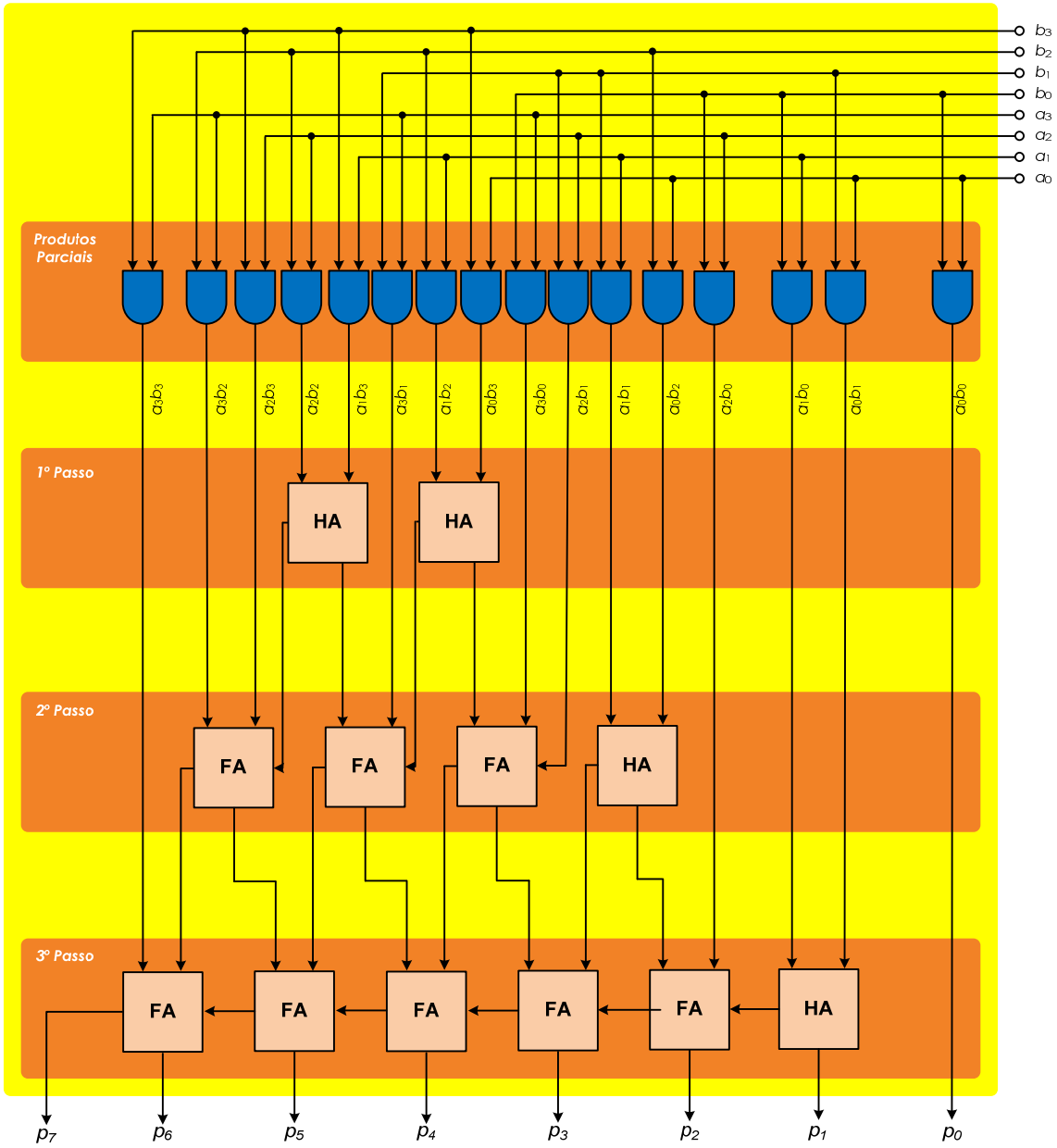


Figura 3.55 – Multiplicador de Wallace com operandos de 4 bits.

3.4.3 Arquitecturas Série

Os multiplicadores matriciais e em árvore têm uma arquitectura paralela na medida em que o cálculo dos produtos parciais é feito em paralelo (portas AND) e a soma desses produtos parciais é dividida em etapas onde múltiplos produtos parciais são somados em paralelo. Esses circuitos são naturalmente rápidos mas em geral requerem uma área para implementação elevada.

Uma alternativa consiste em optar por um circuito mais pequeno mas que no entanto opera em série, sendo os produtos parciais somados sequencialmente. A Figura 3.56 apresenta o diagrama de blocos de um multiplicador Soma-Desloca que usa uma ALU com 2*N* bits em que *N* é o número de bits do multiplicando e do multiplicador.

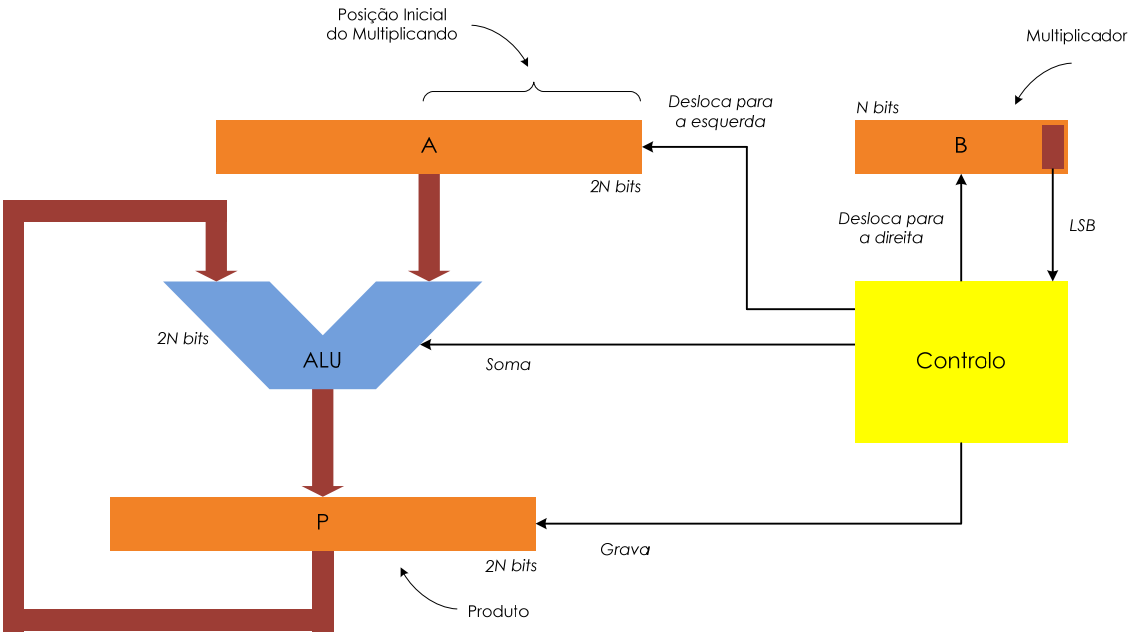


Figura 3.56 – Diagrama de blocos de um multiplicador Soma-Desloca com uma ALU de 2*N* bits.

Inicialmente o multiplicando é carregado na metade direita do registo A e o multiplicador no registo B. A metade esquerda do registo A e o registo do produto são inicializados a 0. A multiplicação ocorre em vários passos. Se o bit menos significativo do multiplicador for 1 então o multiplicando é somado com o valor acumulado no registo P. Se esse bit for 0 nada é feito. De seguida o multiplicando é deslocado para a esquerda de um bit o que corresponde ao deslocamento para a esquerda de uma posição de cada parcela da multiplicação como ilustrado na Figura 3.46. Ao mesmo tempo a palavra guardada no registo B é deslocada para a direita de modo a que o bit mais à direita no registo passa a ser o bit do multiplicador com peso 2. A sequência é então repetida *N* vezes. No fim o resultado da multiplicação encontra-se no registo P.

Existe, no entanto, uma alternativa ao circuito da Figura 3.56 que só necessita de um ALU de *N* bits (em vez de 2*N*) e que usa menos registos. A ideia é aproveitar o registo P para guardar inicialmente o valor do multiplicador. O valor desse registo vai sendo sucessivamente deslocado para a direita de modo a que no fim só contenha o resultado da multiplicação (Figura 3.57).

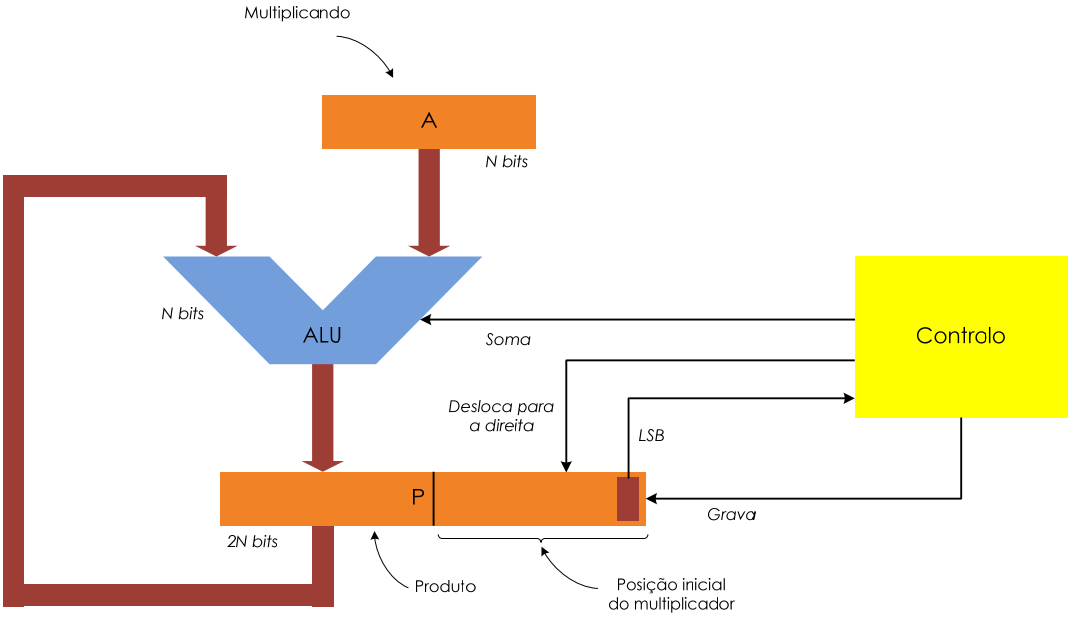


Figura 3.57 – Diagrama de blocos de um multiplicador Soma-Desloca com uma ALU de *N* bits.

O multiplicador soma-desloca só funciona com números inteiros. A multiplicação de números com sinal pode ser realizada adicionando a este circuito a determinação do sinal algébrico do resultado a partir dos bits de sinal dos operadores e a conversão destes da representação em complemento para 2 para uma representação binária. Se a conclusão for a de que o sinal do resultado é negativo (sinais dos operadores diferentes), há que determinar o complemento para 2 do resultado obtido pelo multiplicador soma-desloca (Figura 3.58).

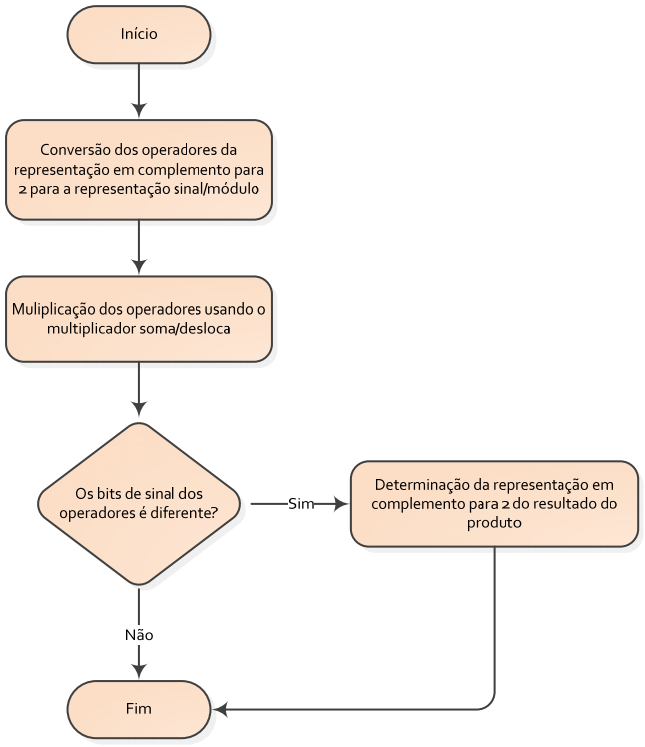


Figura 3.58 – Fluxograma do funcionamento de um multiplicador para cálculo do produtor de dois números com sinal baseado no multiplicador soma-desloca.

3.4.4 Algoritmo de Booth

Como se viu, a multiplicação binária consiste na soma de várias parcelas que podem ser 0 ou o multiplicador deslocado eventualmente para a esquerda. Tendo em conta que a soma de uma parcela cujo valor é 0 não necessita efectivamente de ser efectuada, a multiplicação é tão mais rápida quanto mais parcelas com o valor 0 houver, isto é, quantos mais bits do multiplicador forem 0. O algoritmo de Booth explora esta característica da multiplicação.

Este algoritmo consiste em reescrever o multiplicador como uma diferença de dois números. No caso decimal, por exemplo, é possível escrever 999 como 1000 – 1. Um número que tinha três algarismos diferentes de 0 pode ser escrito como a diferença de dois números que no seu conjunto só têm 2 algarismos diferentes de 0. No caso binário tem-se, por exemplo, que o número 011110 (30_{10}) pode ser escrito como 100000 – 000010 (32_{10}-2_{10}). A sequência de 4 dígitos com o valor 1 foi substituída por dois números que no seu conjunto têm 2 bits iguais a 1. Repare-se como a multiplicação por um número com uma sequência de 1s pode ser substituída pela multiplicação por um número que tem um único 1 à esquerda da sequência de 1s menos a multiplicação por um número que tem um único 1 na posição do dígito mais à direita da sequência (Figura 3.59).

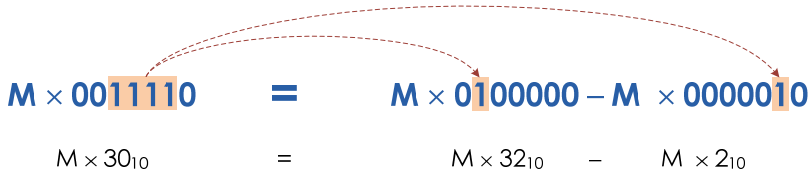


Figura 3.59 – Ilustração da ideia por de trás do algoritmo de Booth – Diminuição do número de 1s do multiplicador.

Isto pode ser generalizado para um número com várias sequências de 1s e mesmo para 1s isolados embora neste último caso haja um aumento do número de 1s (00100 = 01000 – 00100). Estas situações são ilustradas na Figura 3.60.

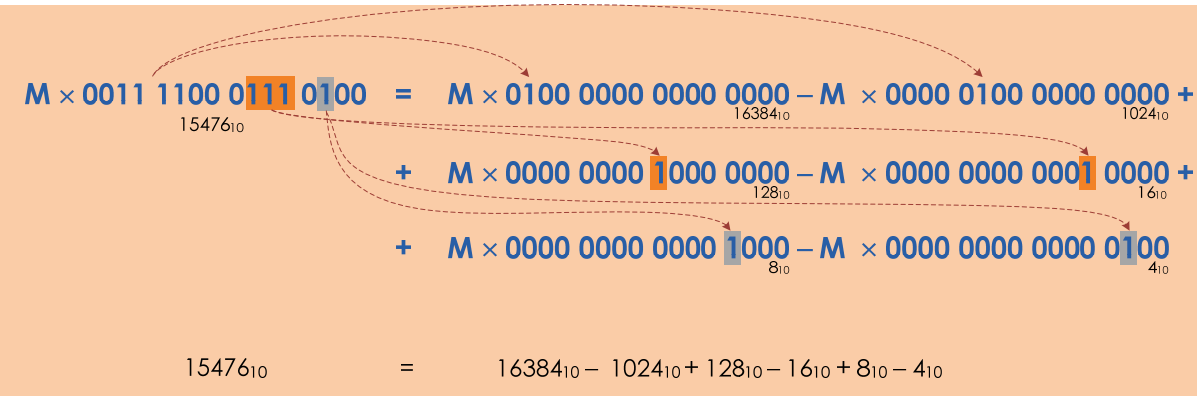


Figura 3.60 – Ilustração do algoritmo de Booth num multiplicador com diversas sequências de 1s.

A técnica acabada de descrever pode ser implementada por um circuito electrónico que faz o "varrimento" do multiplicador, da direita para a esquerda e que subtrai ou soma o multiplicando, devidamente deslocado para a esquerda, a um acumulador consoante encontra o início ou o fim de uma sequência de 1s. A identificação do inicio e fim de uma sequência pode ser feita comparando 2 bits consecutivos durante o varrimento da direita para a esquerda do multiplicador. A existência de um 0 seguido de um 1 assinala o início de uma sequência de uns. A existência de um 1 seguido de um 0 assinala o fim da sequência. Para que isto funcione em sequências "encostadas" à direita do número, junta-se um 0 à direita do multiplicador antes de iniciar o varrimento. Quando os dois bits consecutivos têm o mesmo valor o acumulador não é alterado já que isso corresponde a situações em que o multiplicando seria multiplicado por 0 (meio de uma sequência de 1s ou 0s). A Tabela 3.8 resume as 4 operações que são realizadas quando se encontram as 4 combinações possíveis dos dois bits analisados no algoritmo de Booth de raiz-2.

Tabela 3.8 – Operações realizadas no algoritmo de Booth de raiz-2.

Caso	a_{i+1}	a_i	Operação	Comentários
0	0	0	–	Não é uma sequência de 1's
1	0	1	+ A	Final de sequência de 1's
2	1	0	– A	Início de sequência de 1's
3	1	1	–	Sequência de 1's

A Figura 3.61 apresenta o diagrama de blocos de um multiplicador de Booth de raiz-2. O registo A tem que ter 2*N* bits em que *N* é o número de bits do multiplicando. O multiplicando é colocado nesse registo do lado direito sendo a metade esquerda preenchida com 0. Caso o valor do multiplicando seja negativo é necessário que a representação seja em complemento para 2 e tenha 2*N* bits. A cada passo do varrimento o registo A é deslocado para a esquerda de uma posição. Isso equivale à multiplicação por 2.

O registo B tem *N*+1 bits e contém o multiplicador do lado esquerdo. O bit menos significativo é inicializado com 0. Este registo é deslocado, a cada passo, para a direita de uma posição de modo a analisar os dois bits seguintes do multiplicador. Consoante o valor desses dois bits, de acordo com a Tabela 3.8, nos casos 1 e 2 o módulo de controlo instrui a ALU para somar ou subtrair o valor do registo A (multiplicando deslocado para a esquerda) do valor acumulado no registo P e instrui o registo P para armazenar o resultado proveniente da ALU. Nos outros dois casos nada é feito em termos da ALU, permanecendo o registo P inalterado. O registo P tem 2*N* bits de comprimento pois o produto de dois números de *N* bits pode, no máximo, ter 2*N* bits.

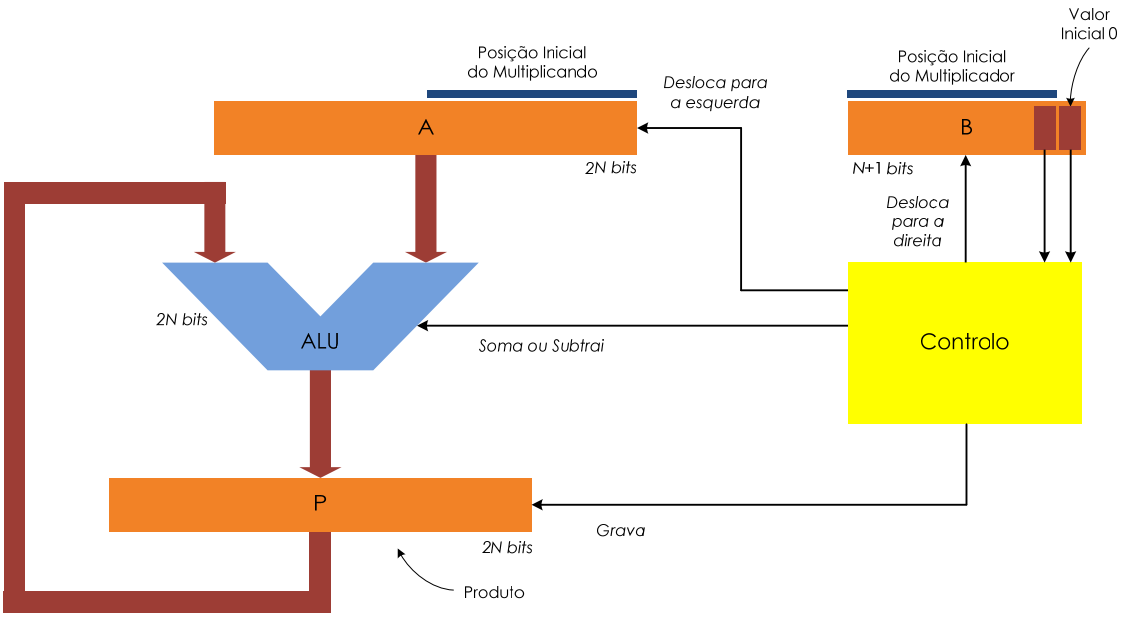


Figura 3.61 – Diagrama de blocos de um multiplicador utilizando o algoritmo de Booth de raiz-2 em que é usada uma ALU de 2*N* bits.

A Figura 3.62 mostra a máquina de estado do controlador usado num multiplicador que usa o algoritmo de Booth de raiz-2.

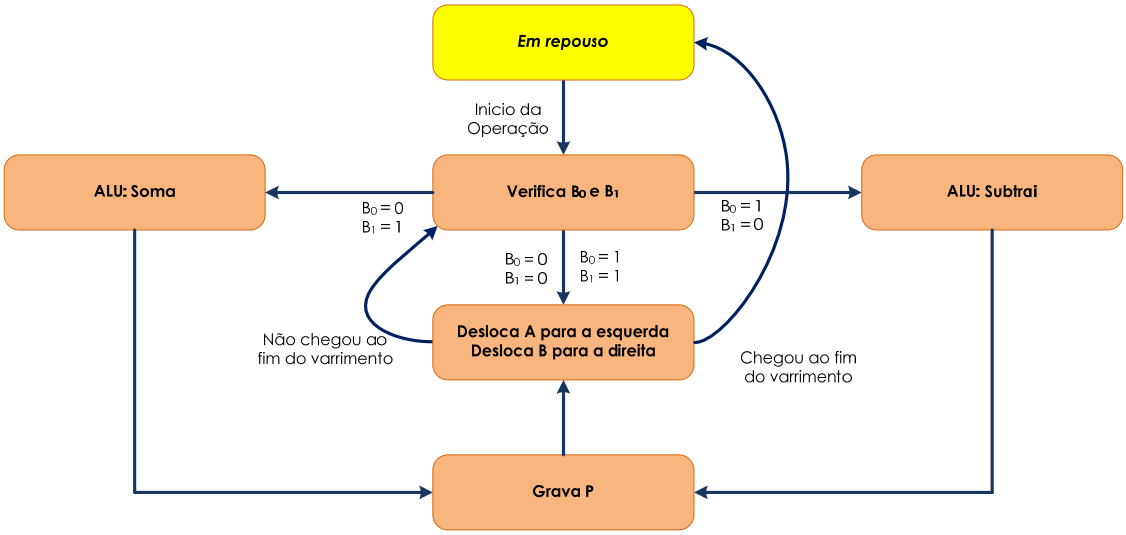


Figura 3.62 – Máquina de estado do controlador de um multiplicador utilizando o algoritmo de Booth de raiz-2.

O multiplicador de Booth funciona mesmo com números negativos representados em complemento para 2, como se ilustra na Figura 3.63.

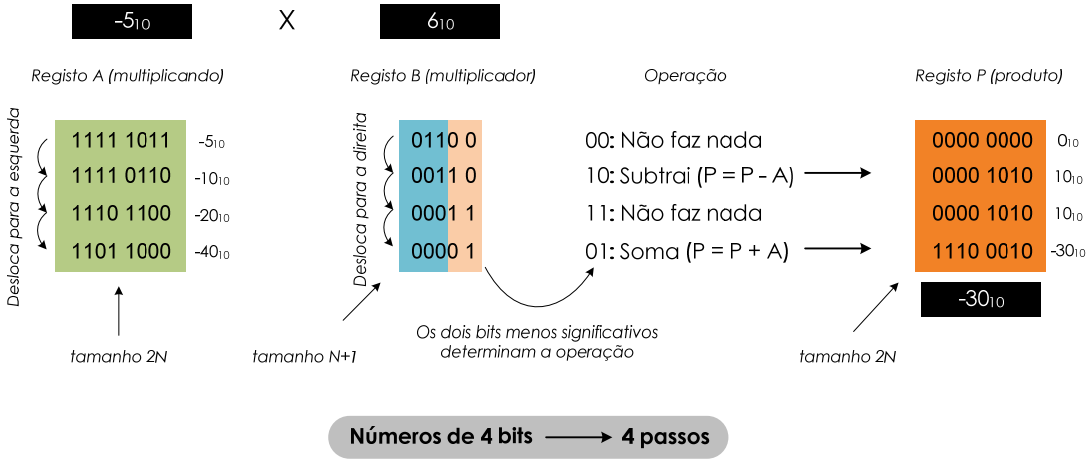


Figura 3.63 – Exemplo de uma multiplicação (-5 x 6) utilizando o algoritmo de Booth de raiz-2 com uma ALU de 2*N* bits.

Tenha-se em atenção que a determinação da representação em complemento para 2 do multiplicando para inicialização do registo A deve ser feita com 2*N* bits (8 bits no caso do exemplo anterior) e não com *N* bits que é o tamanho da palavra a multiplicar.

A Figura 3.64 ilustra o mesmo produto com os operadores trocados – agora o valor negativo é o do multiplicador.

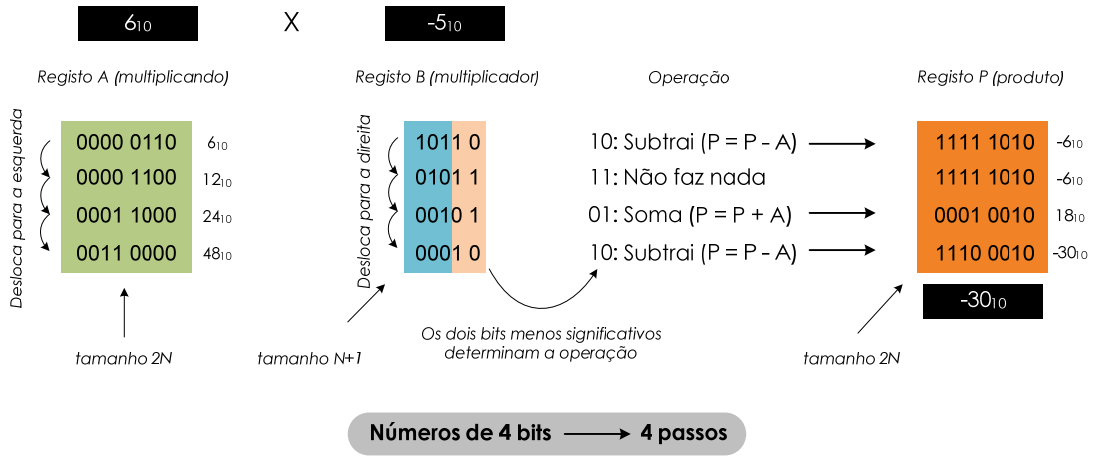


Figura 3.64 – Exemplo de uma multiplicação (6x -5) utilizando o algoritmo de Booth de raiz-2 com uma ALU de 2*N* bits.

Veja-se também os dois casos extremos em que todos os bits do multiplicador são 0 e são 1. No primeiro caso, se todos os bits são 0 e o bit extra que se coloca do lado direito também é zero, nunca se faz nada e portanto o registo P mantém durante todos os passos o valor 0 o que está correcto já que um número vezes 0 deve dar 0. No segundo caso, em que todos os bits do multiplicador são 1 só no primeiro passo é que se tem que fazer alguma coisa pois os dois bits da direita do registo B valem 10. A operação a realizar neste caso é subtrair ao registo P (que inicialmente tem 0) o valor do multiplicando. O resultado é pois o simétrico do multiplicando e esse valor mantém-se durante todos os passos já que os dois bits de controlo vão valer sempre 11. Este resultado é o esperado pois um número binário com representação em complemento para 2 em que todos os bits valem 1 é exactamente -1_{10}. O produto do multiplicando por -1 deve obviamente resultar no simétrico do multiplicando.

A Figura 3.65 ilustra a multiplicação de dois números de 8 bits nomeadamente de 0101 1001 (89_{10}) por 0011 1100 (60_{10}).

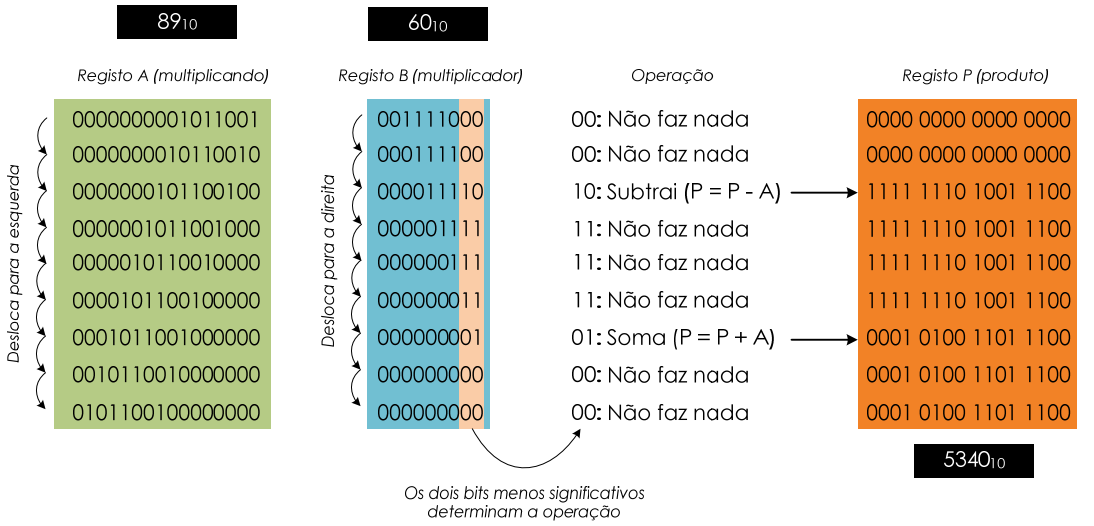


Figura 3.65 – Exemplo de uma multiplicação (89 x 60) utilizando o algoritmo de Booth de raiz-2 com uma ALU de 2*N* bits.

É possível, durante o varrimento do multiplicador, analisarem-se mais do que dois bits de cada vez. Esses algoritmos de Booth modificados designam-se por raiz-m em que o número de bits analisados de cada vez é log(m)+1. O algoritmo de Booth de raiz-4, por exemplo, analisa 3 bits de cada vez. O varrimento, neste caso, é feito em saltos de 2 bits (log(m)).

A Tabela 3.9 apresenta as várias operações que são realizadas consoante o valor dos 3 bits analisados (a_{i+1}, a_i e a_{i-1}). No caso de três 0s (000) ou três 1s (111) o valor do acumulador não é alterado. Se for encontrado um 1 isolado (010) é somado ao acumulador o valor do multiplicando devidamente deslocado para a esquerda (B).

Existem duas hipóteses para a detecção do fim de uma sequência – 001 ou 011. No primeiro caso é somado ao acumulador o valor do multiplicando (devidamente deslocado). No segundo caso é adicionado o dobro do multiplicando pois o bit a seguir ao fim da sequência não é o a_i mas sim o bit à sua esquerda (a_{i+1}) – o deslocamento de um bit para a esquerda representa a multiplicação por 2.

Da mesma forma existem dois casos de início de sequência, nomeadamente o caso 110 e 100. No primeiro caso subtrai-se B e no segundo caso subtrai-se 2B.

Finalmente pode-se encontrar o fim de uma sequência de 1s e o início de uma outra sequência de 1s (101). Nesta situação é subtraído o valor do multiplicando.

Tabela 3.9 – Operações realizadas no algoritmo de Booth de raiz-4.

Caso	a_{i+1}	a_i	a_{i-1}	Operação	Comentários
0	0	0	0	–	Não é uma sequência de 1's
1	0	0	1	+ A	Final de sequência de 1's
2	0	1	0	+ A	1 Isolado
3	0	1	1	+ 2A	Final de sequência de 1's
4	1	0	0	– 2A	Início de sequência de 1's
5	1	0	1	– A	Final de sequência de 1's e início doutra
6	1	1	0	– A	Início de uma sequência de 1's
7	1	1	1	–	Sequência de 1's

O algoritmo de Booth de raiz-4 envolve metade da iterações o que o algoritmo de Booth de raiz-2. Em geral o número de iterações para a multiplicação de palavras de N bits com um algoritmo de raiz-m é de

$$\left\lceil \frac{N}{\log_2 m} \right\rceil . \tag{20}$$

O número de casos distintos é, no entanto, maior (2m).

É também possível implementar o algoritmo de Booth com um multiplicador série em que a unidade aritmética tem N bits em vez de 2N bits, como ilustra a Figura 3.66.

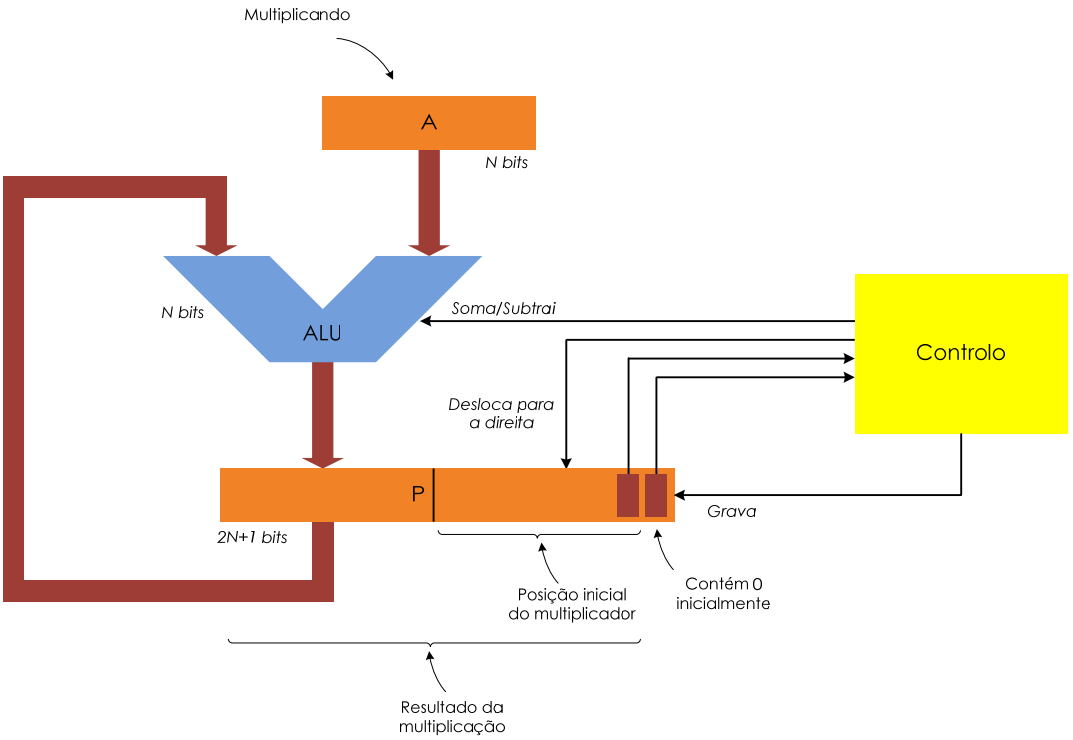


Figura 3.66 – Diagrama de blocos de um multiplicador utilizando o algoritmo de Booth de raiz-2 em que é usada uma ALU de *N* bits.

O multiplicando é carregado no registo A que tem o mesmo número de bits das palavras a multiplicar e o multiplicador é colocado no registo P uma posição afastado do lado direito do registo. Os outros bits desse registo são inicializados com 0.

A Figura 3.67 ilustra a evolução do conteúdo dos registos do produto de -5 por 6 com este multiplicador.

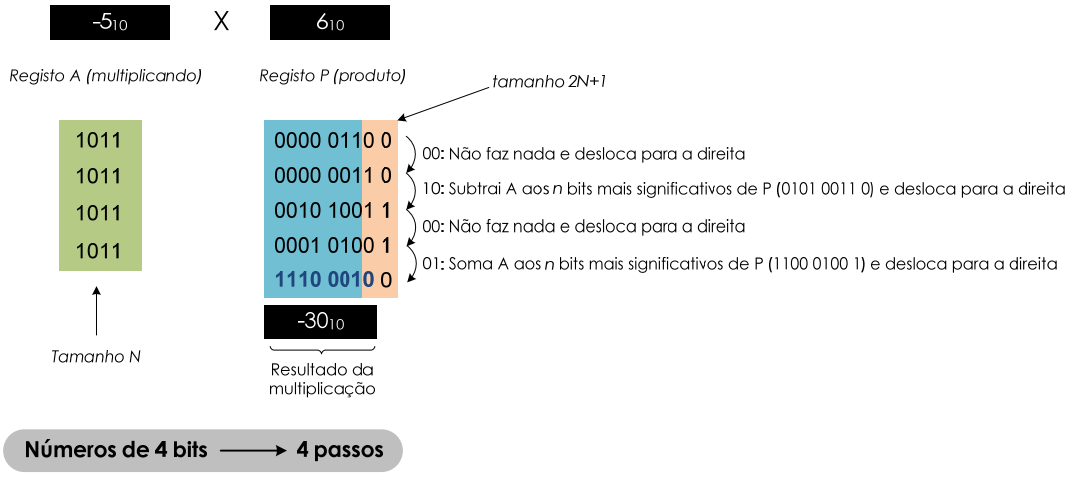


Figura 3.67 – Exemplo de uma multiplicação (-5 x 6) utilizando o algoritmo de Booth de raiz-2 com uma ALU de *N* bits.

Note-se que o deslocamento para a direita do registo P é um deslocamento aritmético, isto é, o bit mais significativo (bit de sinal) mantém o valor (em vez de passar a ser 0 como no caso do deslocamento lógico).

A Figura 3.68 ilustra o mesmo multiplicador a realizar o mesmo produto mas com os operandos trocados.

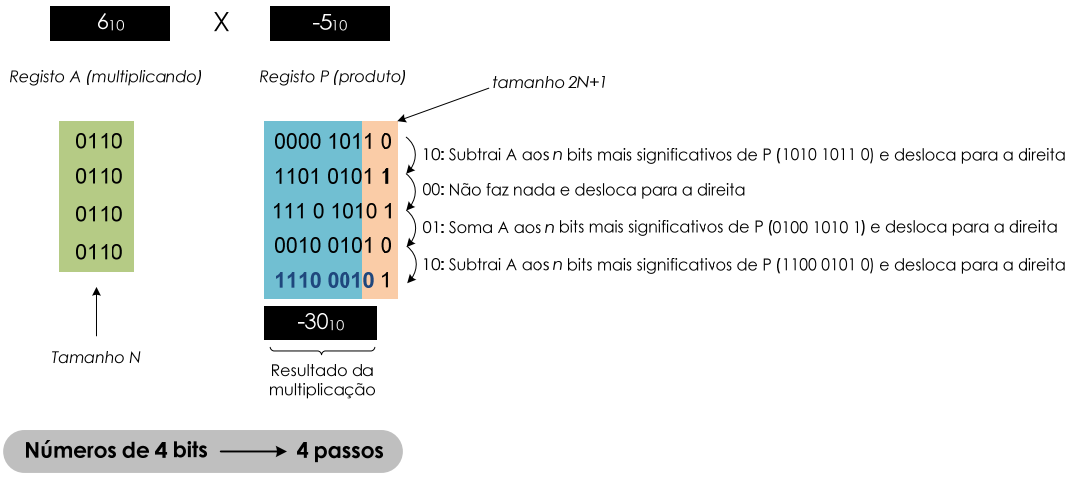


Figura 3.68 – Exemplo de uma multiplicação (6x -5) utilizando o algoritmo de Booth de raiz-2 com uma ALU de *N* bits.

Repare-se que os 4 bits mais significativos do registo P são inicializados com 0 sendo a representação em complemento para 2 do multiplicador efectuada só com 4 bits ao contrário do que acontece com o multiplicando no registo A do multiplicador com a ALU de 2*N* bits (ver Figura 3.63).

3.5 Divisão

A divisão de dois números (dividendo e divisor) consiste em determinar quantas vezes o divisor "cabe dentro" do dividendo (quociente). À parte que sobra dá-se o nome de "resto". Por exemplo, no caso da divisão de 35 por 4 sabemos que divisor (4) cabe 8 vezes no dividendo (35), isto é 4 x 8 = 32, e sobram 3 (resto).

Relembre-se que no caso da multiplicação de dois números de *N* bits dá origem a um resultado que pode requerer 2*N* bits. Assim sendo é usual representar-se, no caso da divisão, o dividendo com 2*N* bits e o divisor com *N* bits. O quociente e o resto são representados por *N* bits. A divisão de dois números com 2*N* e *N* bits respectivamente pode, no entanto requerer mais do que *N* bits, ou seja, mais do que é possível representar. É necessário detectar esses casos e activar uma indicação de que houve *overflow*.

Imagine-se que *N* = 4. Se, por exemplo, o dividendo for 90 (0101 1010) e o divisor for 2 (0010) o quociente é 45 (0010 1101) e o resto 0. O número 45 requer mais do que 4 bits para ser representado o que é mais do que o disponível para o efeito. O caso limite é quando o divisor é 0. Nesse caso o quociente seria infinito.

Isto acontece sempre que o dividendo for maior ou igual a 2^N vezes o divisor.

A forma mais simples de determinar o quociente e o resto seria ir subtraindo o divisor do dividendo, repetidamente, até que o resto fosse menor do que o divisor. O número de vezes que fosse possível fazer essa subtracção seria o valor do quociente (Figura 3.69).

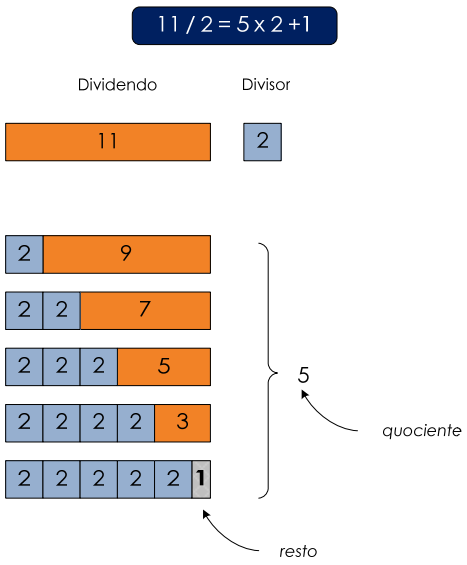


Figura 3.69 – Ilustração de um procedimento de divisão que envolve a subtracção sucessiva do divisor.

Este processo é muito ineficiente pois pode requerer um número muito elevado de subtracções. Não é usado nem mesmo quando queremos fazer a divisão usando papel e lápis. A técnica que é costume usar consiste em subtrair do dividendo múltiplos do divisor cada vez mais pequenos (em vez de subtrair sempre o mesmo valor do divisor). Isso tem a ver com a forma como representamos os números – usando algarismos que correspondem a múltiplos da unidade. O número 142, por exemplo, representa a soma de 1 centena com 4 dezenas e 2 unidades. Assim, quando estamos a efectuar a divisão de um número com 3 algarismos, começamos por subtrair centenas de divisores até o resto ser menor do que 100 vezes o divisor. De seguida fazemos o mesmo com dezenas de divisores e finalmente com divisores até obtermos um valor menor do que o divisor (o resto). Este procedimento está ilustrado na Figura 3.70.

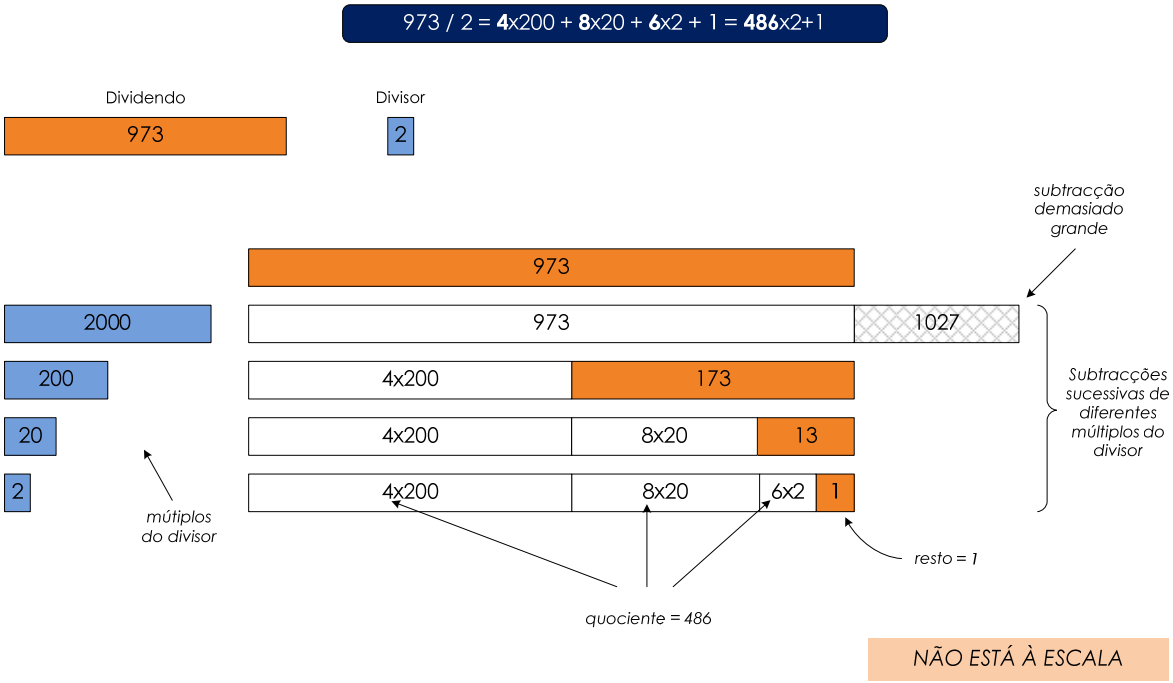


Figura 3.70 – Ilustração de um procedimento de divisão que envolve a subtracção sucessiva de múltiplos do divisor.

A Figura 3.71 ilustra a forma comum de fazer a divisão usando papel e lápis com base na subtracção do dividendo de múltiplos do divisor (unidades, dezenas, centenas, etc.). Esse procedimento começa com o maior múltiplo do divisor que é menor do que o dividendo. No caso do exemplo da Figura 3.71 esse múltiplo é o correspondente às centenas. A primeira subtracção feita, "9 menos 8", é na verdade a subtracção "973 menos 800" mas os dois algarismos da direita são ignorados (para já). O passo seguinte diz respeito às dezenas. Ao resto da subtracção anterior (173) é subtraído 80 x 2 (160). Da mesma forma o algarismo da direita não é usado explicitamente, ficando 17 - 16.

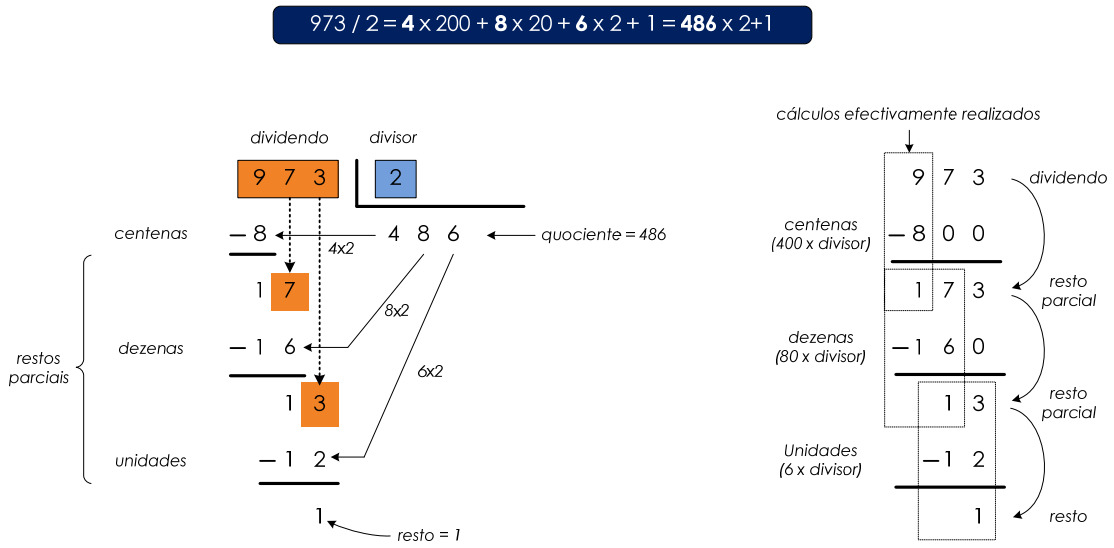


Figura 3.71 – Ilustração da divisão decimal de números inteiros usando papel e lápis.

Note-se que cada algarismo do quociente é determinado "de cabeça" com base no valor do divisor e do dividendo (ou restos intermédios). No caso da Figura 3.71, por exemplo, o algarismo mais significativo do quociente (4) é determinado a partir do algarismo 9 do dividendo e do valor do divisor (2). "De cabeça" obtém-se que o divisor deve ser multiplicado por 4 para se obter o maior número que ainda é menor do 9 (neste caso o 8). Essa divisão "de cabeça" é, em geral, mais simples que a divisão completa pois envolve números menores. Quanto mais prática, mais rápida é essa determinação. O processo mental levado a cabo passa muitas vezes por testar diversas hipóteses para o algarismo.

3.5.1 Arquitectura em Série

No caso dos computadores, no entanto, não existe um conhecimento à priori de qual será o melhor valor para o algarismo. As possibilidades são testadas começando em 9 e indo até 0 fazendo-se o produto desse algarismo pelo divisor e verificado se é menor do que o resto parcial ou não. No caso do algarismo mais significativo do quociente da Figura 3.71, 9 x 2 = 18 é maior do que 9, 8 x 2 = 16 é maior do que 9, e assim sucessivamente até se chegar a 4 x 2 = 8 que já é menor do que 9.

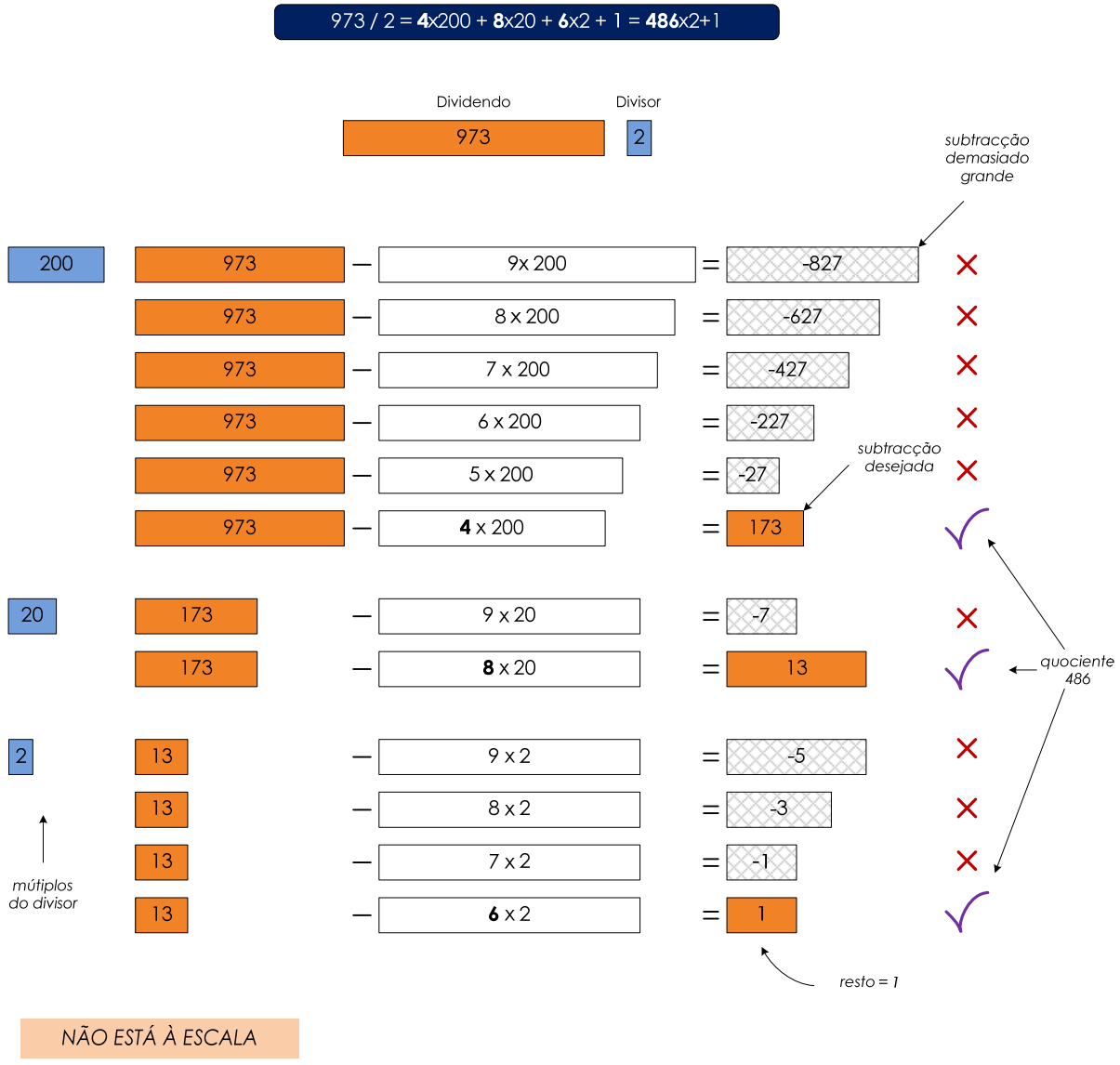


Figura 3.72 – Ilustração de um procedimento de divisão que envolve a subtracção sucessiva de múltiplos do divisor.

Na verdade os computadores não usam representação em base 10 mas sim em base 2 o que torna esse procedimento muito mais simples pois cada algarismo do quociente só pode assumir dois valores: 0 e 1. Por outro lado, os múltiplos do divisor usados não são potências de 10 (10, 100, 1000, etc.), mas sim potências de 2 (2, 4, 8, 16, etc.). A Figura 3.73 exemplifica o caso da divisão de 13 por 5 usando representação em base 2 embora a indicação dos valores seja feita em base 10.

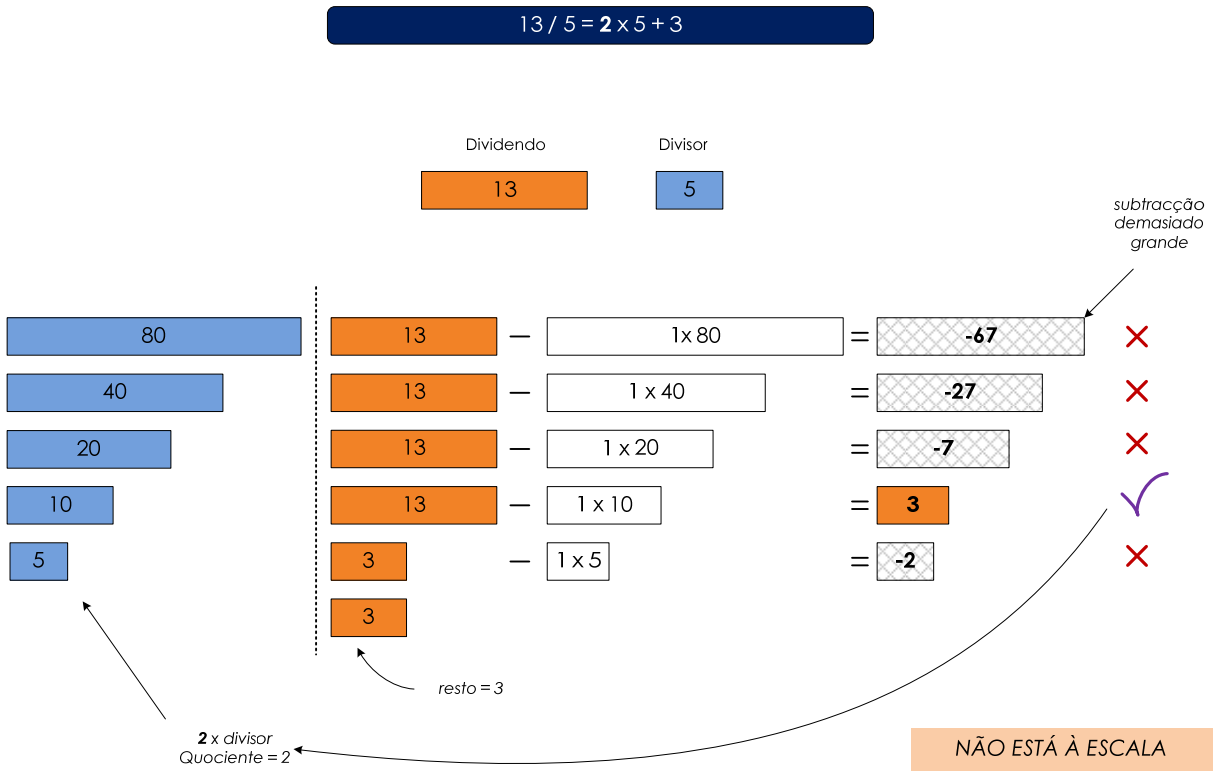


Figura 3.73 – Ilustração de um procedimento de divisão de dois números em base 2 mas onde é usada base 10 para indicar os valores.

O mesmo exemplo é apresentado na Figura 3.74 indicando-se agora os valores em base 2 bem como o nome dos registos utilizados na arquitectura apresentada na Figura 3.76.

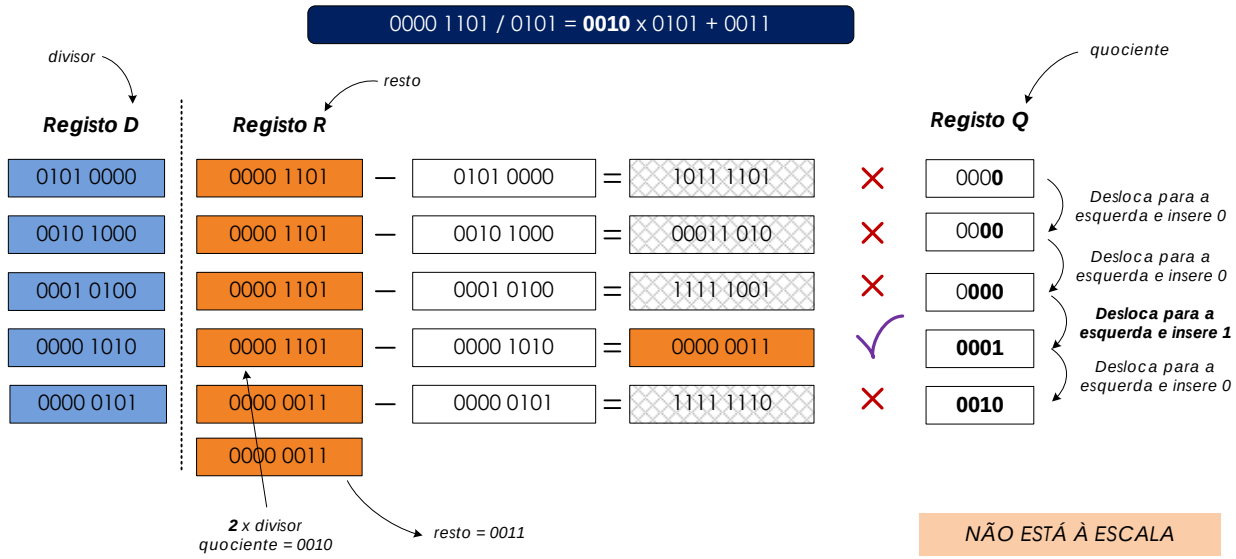


Figura 3.74 – Ilustração de um procedimento de divisão de dois números em base 2.

Na realidade o resultado de cada subtracção é colocado no registo R. Caso se verifique que é negativo é necessário, antes do passo seguinte voltar a colocar nesse registo o valor que tinha antes. Isso é conseguido voltando a somar o divisor (registo D) ao conteúdo do resto (registo R). Este pormenor é tornado explicito na Figura 3.75.

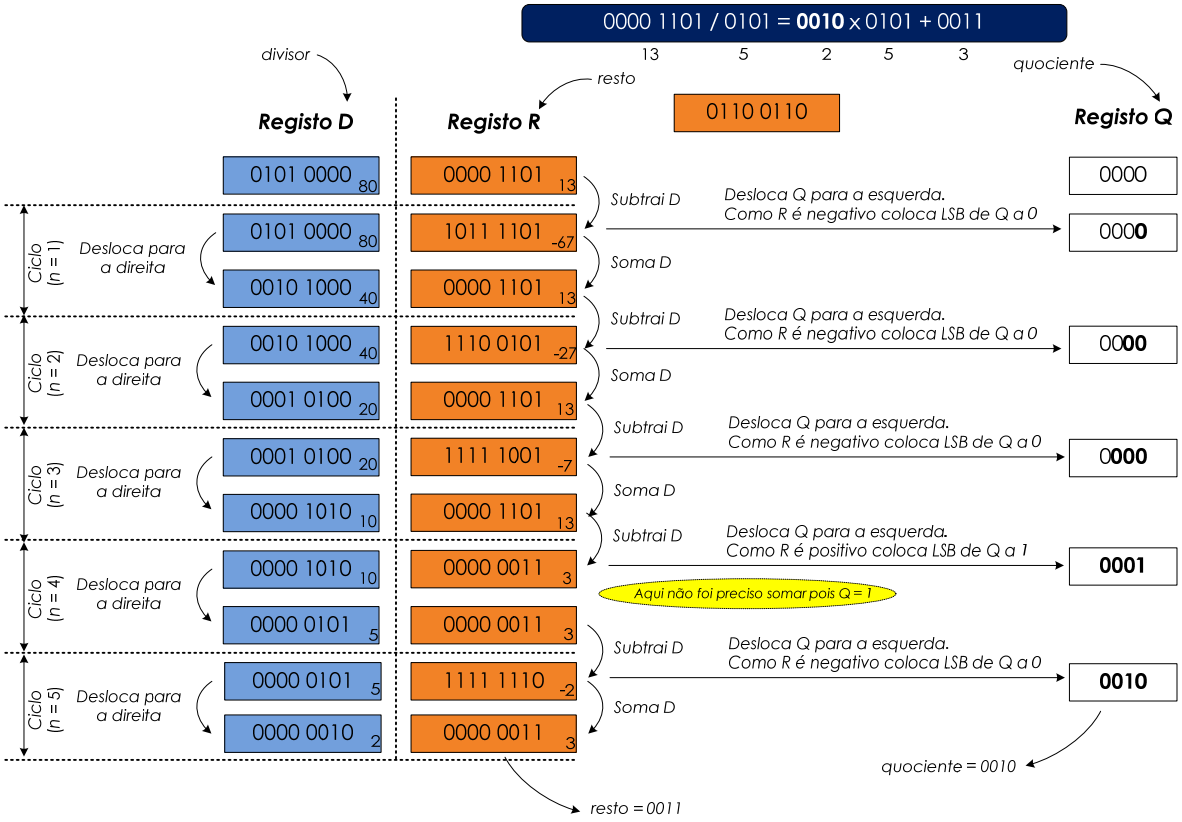


Figura 3.75 – Ilustração da evolução dos registos num divisor Subtrai-Desloca com uma ALU de 2*N* bits.

Para implementar este algoritmo em hardware é preciso uma ALU para somar e subtrair, 3 registos para guardar o divisor (D), o quociente (Q) e os restos parciais (R) e uma unidade de controlo. A unidade de controlo recebe indicação da ALU se o resultado da soma/subtração é positivo ou não e controla os sinais de deslocamento de registo, selecção de operação na ALU e escrita de 0 ou 1 no LSB no registo Q. Estes circuitos são interligados como ilustra a Figura 3.76.

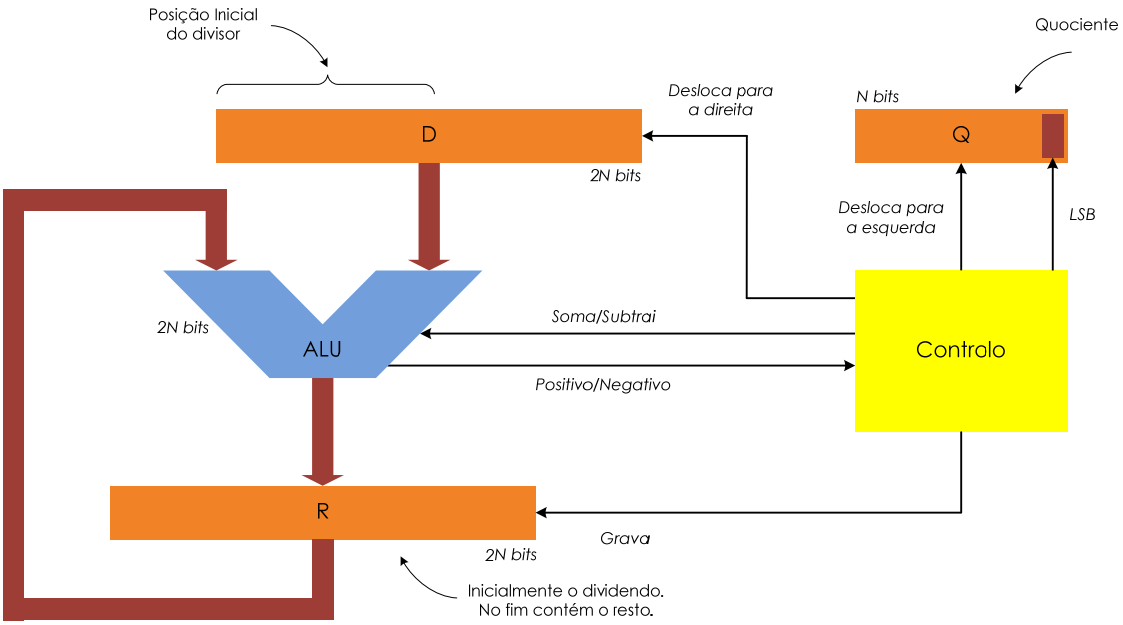


Figura 3.76 – Diagrama de blocos de um divisor Subtrai-Desloca com uma ALU de 2*N* bits.

A unidade de controlo é implementação de uma máquina de estados finita com os estados descritos na Figura 3.77.

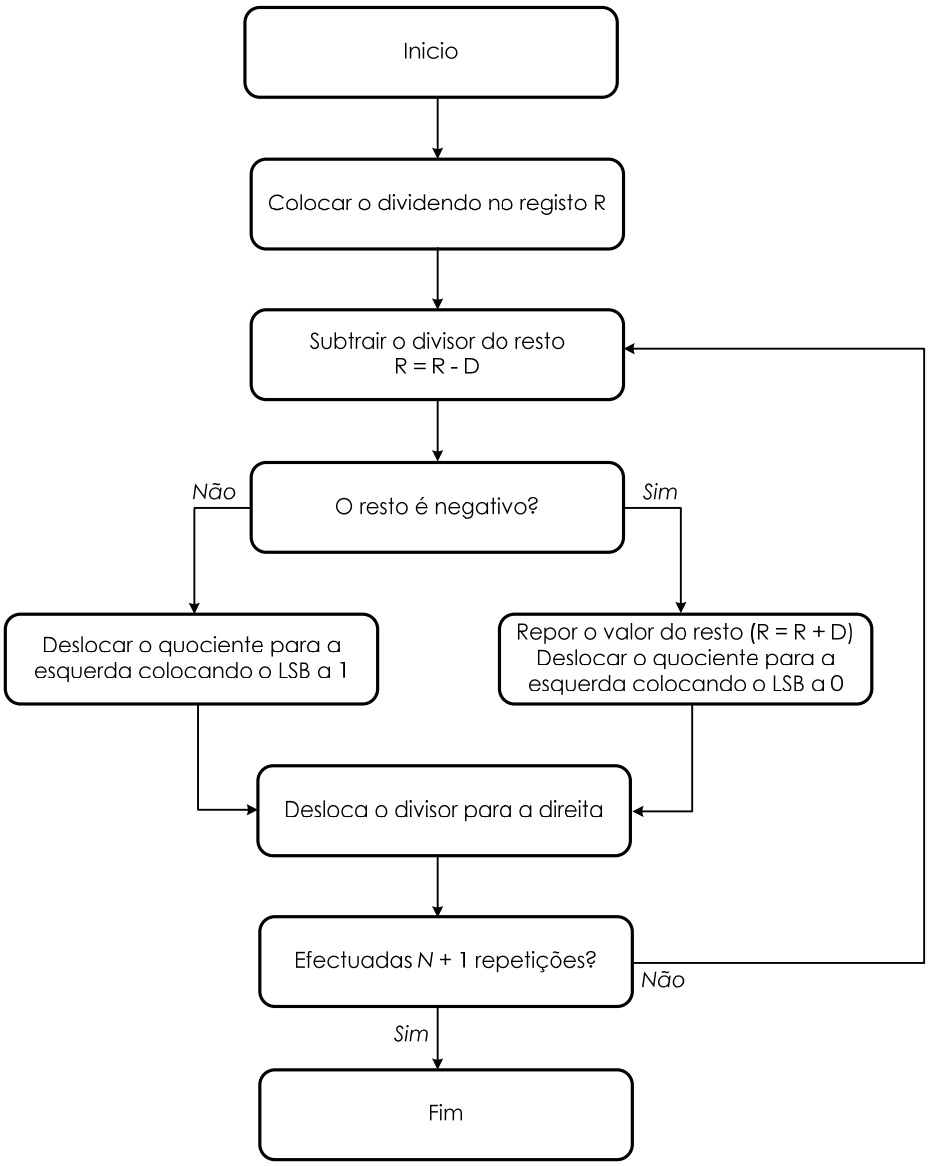


Figura 3.77 – Fluxograma do divisor Subtrai-Desloca com uma ALU de 2*N* bits.

Também com o caso da divisão é possível usar uma arquitectura semelhante mas em que é usado uma ALU de *N* bits em vez de 2*N* bits (Figura 3.78).

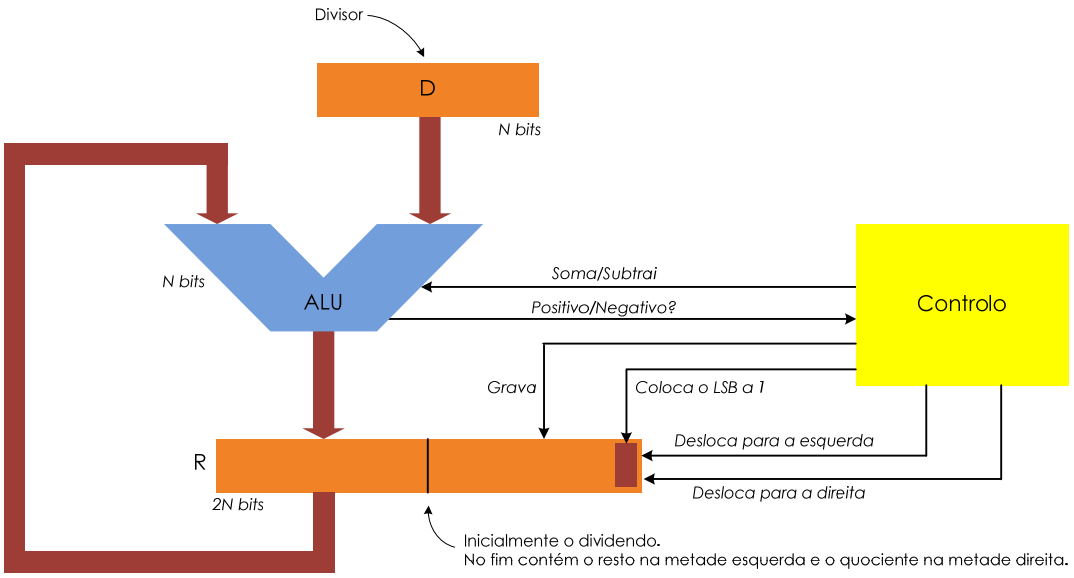


Figura 3.78 – Diagrama de blocos de um divisor Subtrai-Desloca com uma ALU de *N* bits.

A máquina de estados finita da unidade de controlo é, neste caso, a ilustrada na Figura 3.79.

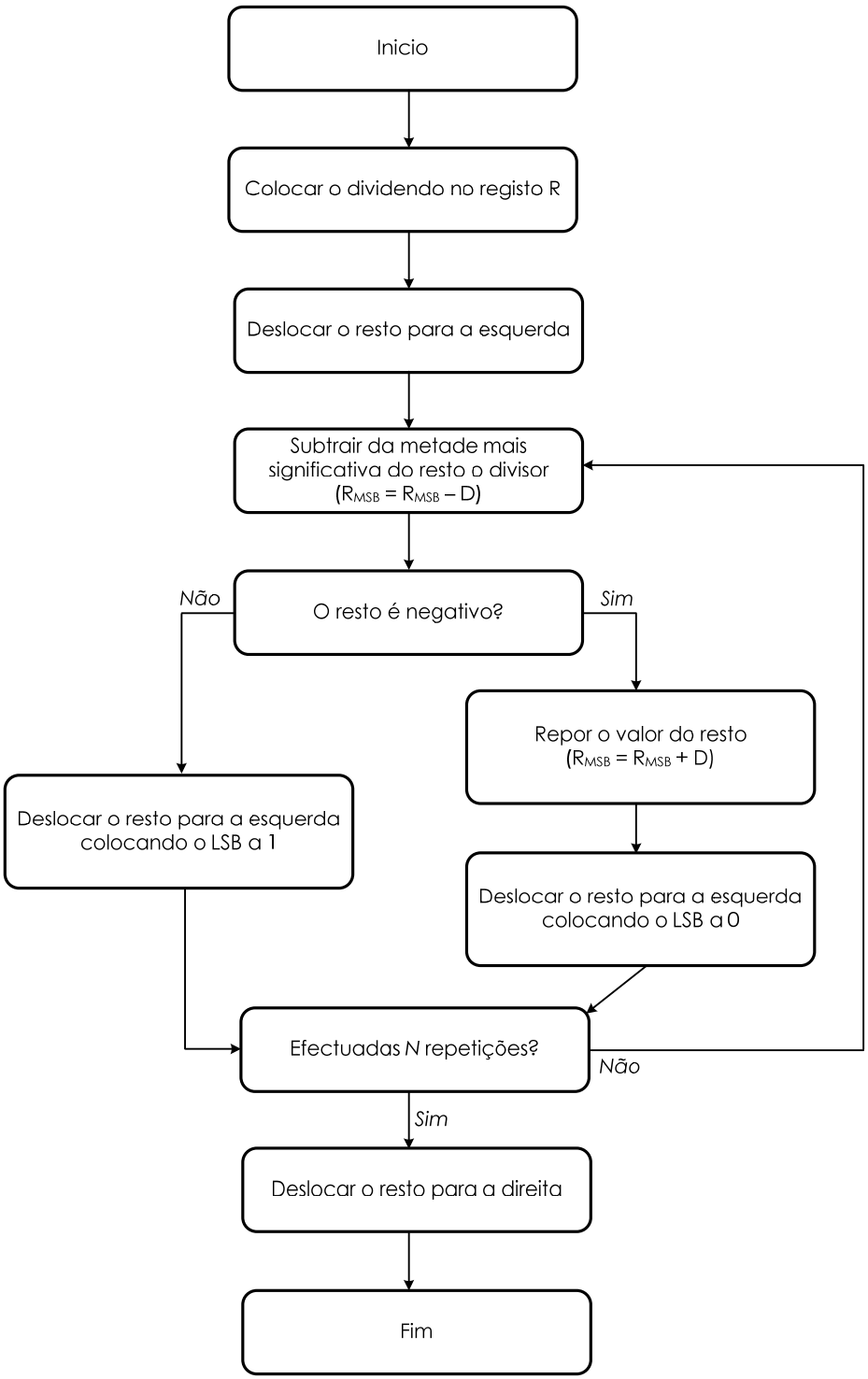


Figura 3.79 – Fluxograma do divisor Subtrai-Desloca com uma ALU de *N* bits.

A Figura 3.80 mostra um exemplo da evolução dos registos num divisor Subtrai-Desloca com ALU de *N* bits para o caso concreto em que *N* = 4.

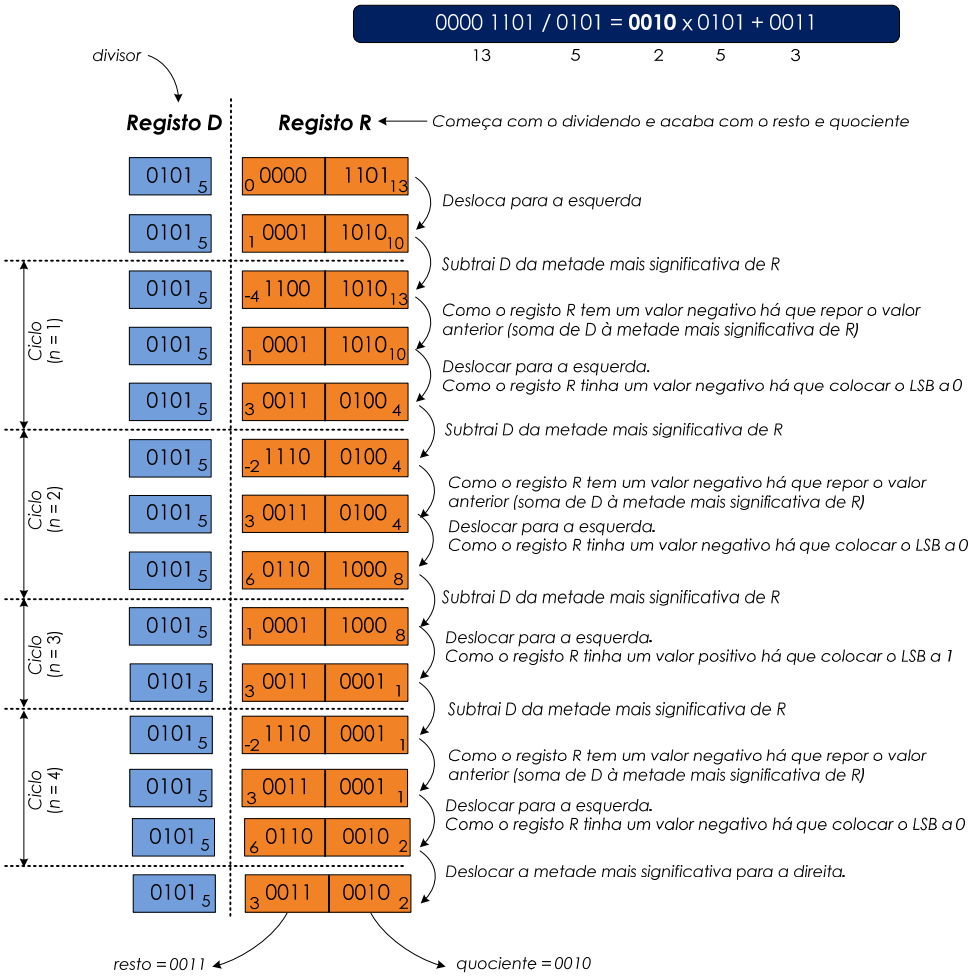


Figura 3.80 – Ilustração da evolução dos registos num divisor Subtrai-Desloca com uma ALU de *N* bits.

3.5.2 Algoritmo SRT

O algoritmo SRT é um algoritmo muito popular cujo nome advém das iniciais de 3 pessoas que o criarem de forma independente praticamente em simultâneo (Sweeney, Robertson e Tocher). Este algoritmo consegue obviar uma grande desvantagem do algoritmo usado no divisor Subtrai-Desloca: são realizadas operações matemáticas (somas e subtracções) que à posteriori percebe-se que não eram necessárias. Isto acontece porque para determinar cada dígito do quociente (q_i) é necessário efectuar uma subtracção (resto parcial - divisor). Caso o resultado dessa subtracção seja positivo então o valor correcto de q_i é 1 e o resto parcial passa a ser o resultado dessa subtracção. Até aqui não há nenhum desperdício (a subtracção realizada era mesmo necessária). No entanto, se o resultado da subtracção for negativo, então o valor correcto de q_i é 0 e o resto parcial mantém-se. A subtracção efectuada foi então um desperdício já que o seu resultado é descartado. Na verdade, o resultado da subtracções foi logo escrito no registo R sendo preciso voltar a colocar aí o valor anterior. Nos exemplos dados anteriormente a reposição era feita voltando-se a somar ao registo R o valor do divisor que tinha sido usado para efectuar a subtracção. Outra alternativa seria guardar o conteúdo do registo R em outro registo temporário antes da subtracções por forma a efectuar o recuperação do valor do resto parcial a partir desse registo temporário. Seja como for nos casos em

que $q_i = 0$ há subtracções que foram realizadas desnecessariamente. O algoritmo SRT evita isso como se verá de seguida.

A subtracção e teste do sinal do resultado realizado pelo divisor Subtrai-Desloca serve para determinar se o resto parcial é maior ou menor do que o divisor (na verdade um múltiplo do divisor). Não é possível, em geral, comparar a magnitude de dois números sem usar todos os seus algarismos. Isso é, no entanto possível em alguns casos particulares. Por exemplo, para comparar os números 1001 (9_{10}) e 0011 (3_{10}) basta olhar para o bit mais significativo (neste caso usou-se notação sem sinal). Se um dos número tiver esse bit igual a 1 e ou outro número tiver esse bit igual a 0, como acontece neste exemplo, então o número maior é aquele cujo bit mais significativo é 1. Se os bits mais significativos dos dois números forem iguais então é preciso olhar para o bit seguinte (à direita) e decidir com base nele o que pode não ser possível se forem iguais. Em certos casos pode ter que se analisar todos os bits até ao menos significativo para encontrar uma diferença e se possível decidir qual o maior. Obviamente quantos mais bits tiverem que ser analisados mais tempo demora a comparação. No pior caso, quando todos têm que ser analisados, o tempo demorado é o mesmo do que necessário para subtrair um de outro.

O divisor SRT faz, neste aspecto, um compromisso. Contenta-se em analisar só dois bits mais significativos de um número para decidir se é o resto parcial ou o divisor que é maior. Este procedimento é muito mais rápido mas não dá uma resposta definitiva. Dito de outro modo, o divisor SRT faz uma estimativa de cada bit de quociente tendo em conta só o valor dos dois dígitos mais significativos do resto parcial. Nem chega a olhar para os bits do divisor como se verá.

A estimativa de uma dado bit de quociente é usada então para calcular o novo resto parcial. Se a estimativa do bit de quociente for 0 ($q_i = 0$) o resto parcial mantém-se inalterado (nenhuma subtracção é efectuada tal como no caso do divisor Subtraí-Desloca). Como se explicará adiante o divisor nunca se engana quando estima $q_i = 0$. Se a estimativa do bit de quociente for 1, por exemplo, o divisor é subtraído do resto parcial. Se essa estimativa estiver certa (algo que o circuito não sabe) o resto parcial era de facto maior do que o divisor e portanto, depois da subtracção, ficou menor do que o divisor (como é suposto acontecer). No entanto, se a estimativa $q_i = 1$ estiver errada, isto é, se o resto parcial afinal for inferior ao divisor, o resultado da subtracção (que não devia ter sido feita mas que foi) ficará negativo. Numa divisão normal (feita a papel e lápis, por exemplo) isto nunca acontece. No caso do divisor SRT isto é uma consequência de se usar um valor estimado de q_i e não o valor correcto devido a querer-se decidir o seu valor mais rapidamente. O divisor SRT está, no entanto preparado para que isto aconteça.

Um valor negativo para o resto parcial significa que foi retirado do resto parcial anterior mais do que se devia ter tirado (foi tirado o divisor quando não se devia ter tirado nada). Nesta altura já está disponível informação sobre o facto de o bit de quociente correcto nesse caso ser 0 e não 1 como foi estimado (basta olhar para o sinal algébrico do resto parcial). O divisor SRT não tenta voltar a trás mudando o valor de q_i e corrigindo o resto parcial com a reposição do valor anterior como faz o divisor Subtrai-Desloca. O divisor SRT mantém esse resto parcial negativo e delega para a etapa

seguinte (determinação do bit de quociente à direita, menos significativo) a correcção desse resto parcial negativo.

A correcção de um resto parcial negativo é conseguida acrescentando uma terceira hipótese à estimativa de um bit de quociente. Em vez de poder ser só 0 ou 1, esse bit pode ter também um outro valor que às vezes é representado por "-1" outra vezes por algo como "$\bar{1}$". Pode-se encarar esses três símbolos, "0", "1" e "$\bar{1}$" como representando três hipóteses diferentes e não necessariamente 3 algarismos que constituem um número (o quociente da divisão). A cada bit do quociente é atribuído um desses 3 símbolos ao longo do desenrolar do calculo obtendo-se no fim algo como "$010\bar{1}\ 1\bar{1}00$". Isto não representa um número binário. Representam só as estimativas efectuadas na determinação de cada bit do quociente. Numa última etapa do funcionamento do divisor SRT essa representação é transformada num número binário.

Essa terceira hipótese para um dado bit de quociente ("$\bar{1}$") significa que o resto parcial é demasiado baixo (muito negativo) e que portanto deve agora ser somado o divisor ao resto parcial para corrigir esse facto (em vez de não fazer nada, quando q_i = 0 ou subtrair quando q_i = 1). Note-se que nesta etapa o peso do divisor em relação ao resto parcial é menor do que na etapa anterior. Isto porque em cada etapa o divisor é dividido por dois. Na prática, por forma a facilitar a implementação, o resto parcial é que é multiplicado por 2 entre cada etapa. O resultado é, no entanto o mesmo: o valor que é subtraído ao resto parcial em cada etapa é cada vez menor. Por esta razão uma determinada estimativa errada numa dada etapa pode não ser corrigida totalmente na etapa seguinte. Depende de quão negativo ficou o resto parcial aquando da estimativa errada do bit do quociente.

Por forma a tornar esta explicação mais clara vai-se apresentar um exemplo numérico. Antes disso, no entanto, há que apresentar alguns detalhes sobre a implementação do divisor SRT. Em primeiro lugar há que explicar como é que é possível estimar um bit de quociente olhando só para dois bits do resto parcial e sem olhar para nenhum bit do divisor.

Antes de se começar o procedimento de determinar os bit do quociente, o circuito altera o valor do divisor (D) de modo a que ele tenha um valor igual ou maior do que 0,5 e menor do que 1 (*normalização do divisor*). Isso é feito em duas fases: 1) ajuste do sinal e 2) escalamento da magnitude. Caso o sinal for negativo é calculado o simétrico do divisor e do dividendo (fazendo o complemento para 2) tornando-se esses valores o novo divisor e dividendo respectivamente. O sinal do resultado obviamente ficará inalterado. A segunda fase desta normalização consiste em eliminar os dígitos mais significativos que tenham valor 0. Isso consegue-se fazendo um deslocamento para a esquerda do conteúdo do registo onde está armazenado o divisor (D) até que o bit mais significativo seja 1. Considere que o número de deslocamentos efectuados é k. Essa operação consiste no fundo em multiplicar o divisor por uma potência de 2 (2^k). Desloca-se então o dividendo também k posições para a esquerda de modo a manter o mesmo quociente da divisão. Por exemplo, imagine-se que se pretendia efectuar a divisão de d por D de modo a determinar o quociente q e o resto r tal que

$$d = D \times q + r \,. \tag{20}$$

Se em vez desta divisão se efectuar a divisão de 4d por 4D obter-se-á um valor de resto diferente (*r*'):

$$(4d) = (4D) \times q + r' . \tag{20}$$

Para obter-se os valores desejados de *r* basta dividir r' por 4, ou seja,

$$r = \frac{r'}{4} . \tag{20}$$

Isto pode-se provar inserindo (20) em (20) e verificando que obtém-se (20).

Resumindo, no fim do algoritmo, quando *r*' for conhecido, basta deslocar o resto *k* posições para a direita (dividindo por 2^k).

Na segunda fase de normalização é também acrescentado um separador decimal à esquerda do bit mais significativo (que nesta altura já é 1). Este acrescento do separador decimal é uma operação conceptual para melhor se compreender o algoritmo. A implementação em hardware não usa nada relacionado com separadores decimais.

Depois desta normalização o divisor tem o seguinte formato: ",1xxx" em que "x" representa um algarismo que pode valer 0 ou 1. Foram usados só 4 bits como forma de exemplo. O divisor pode ter um número de algarismos qualquer. Com este formato o menor número que se pode ter é ",1000" que é equivalente a 0,5 em base 10. Por outro lado, o maior número que se pode ter é ",1111" que é exactamente o maior número possível antes do número 1 (considerando números com 4 bits). O divisor ficou assim normalizado entre 0,5 e 1.

De seguida é necessário normalizar o dividendo. Note-se que o dividendo é o valor que inicialmente o resto parcial assume. O dividendo é normalizado por forma a ser maior ou igual a -0,5 e menor ou igual a 0,5. Isso vai fazer com que o dividendo seja sempre menor do que o divisor. Tanto o dividendo como os restos parciais podem ser números negativos. Têm portanto um bit de sinal e são representados em notação de complemento para 2. A normalização consiste em deslocar o dividendo para a direita uma vez se os dois bits mais significativo forem diferentes mantendo o bit mais significativo (bit de sinal),

$$\frac{d}{2} = D \times q' + r' . \tag{20}$$

O número 01xx passara a 001x e 10xx passará a 110x. Se isto for feito, no fim do algoritmo obtém-se valores de quociente (*q*') e resto (r') que precisão ser corrigidos multiplicando cada um por dois, ou seja, deslocando o conteúdo dos seus registos de uma posição para a esquerda.

$$q = 2q' \quad \text{e} \quad r = 2r' . \tag{20}$$

Finalmente coloca-se um separador decimal à direita do bit mais significativo (bit de sinal).

Depois destas duas normalizações feitas o dividendo será inferior, em módulo, ao divisor. O mesmo acontecerá com o resto parcial ao longo da determinação dos bits do quociente.

É possível agora introduzir a forma como é feita a estimativa de um dado bit do quociente. O divisor nunca muda ao longo de todo o procedimento. É sempre igual ou superior a 0,5. Os restos parciais, por seu lado, alteram-se pois a eles é somado ou subtraído o divisor e entre cada passo é multiplicado por dois. A estimativa é efectuada olhado só para os dois bits mais significativos do resto parcial. Há então 4 possibilidades:

1) Se o resto parcial for positivo e menor do que 0,5 então é menor do que o divisor e o bit de quociente estimado é 0. Isto acontece quando os dois bits mais significativos são 0 (**0,0xxxxxx**).
2) Se o resto parcial for maior ou igual a 0,5 ele pode ou não ser maior do que o divisor pois este está entre 0,5 e 1. Esta condição é detectada se os bits mais significativos forem 0 e 1 (**0,1xxxxxx**). Nesta caso a estimativa é que o resto é maior do que o divisor e portanto o bit de quociente é 1. Essa estimativa pode estar errada. Olhando só para os dois bits mais significativos do resto parcial e do divisor é impossível saber.
3) Se o resto parcial for menor do que -0,5 (**1,0xxxxxx**) quer dizer que a estimativa anterior do bit de quociente estava errada. O resto parcial é obviamente menor do que o divisor. A ele deve ser somado o divisor para corrigir esta situação. A hipótese escolhida para o bit de quociente é a $\bar{1}$.
4) Se o resto parcial for negativo mas maior do que -0,5 (**1,1xxxxxx**) então também houve um erro de estimativa num dos bits de quociente anteriores. A opção usual, no entanto é de não corrigir o resto parcial nesta altura. A correcção possível seria soma o divisor ao resto parcial. Como o divisor tem um valor entre 0,5 e 1 essa correcção podia levar o novo resto parcial a ter um valor maior do que 0,5 (-0,1 + 1 = 0,9 por exemplo) o que é demasiado. Nos passos seguintes, em que o peso relativo do divisor é menor, será mais adequado fazer a correcção e portanto neste caso a estimativa é $q_i = 0$.

A estimativa do bit de quociente a partir do resto parcial do passo anterior é feita do seguinte modo:

$$q_i = \begin{cases} 1 & \text{se} \quad 2r_{i-1} \geq 0,5 \\ 0 & \text{se} \quad -0,5 \leq 2r_{i-1} < 0,5 \\ \bar{1} & \text{se} \quad 2r_{i-1} \leq -0,5 \end{cases} . \tag{20}$$

A Figura 3.81 mostra o fluxograma do procedimento usado no divisor SRT.

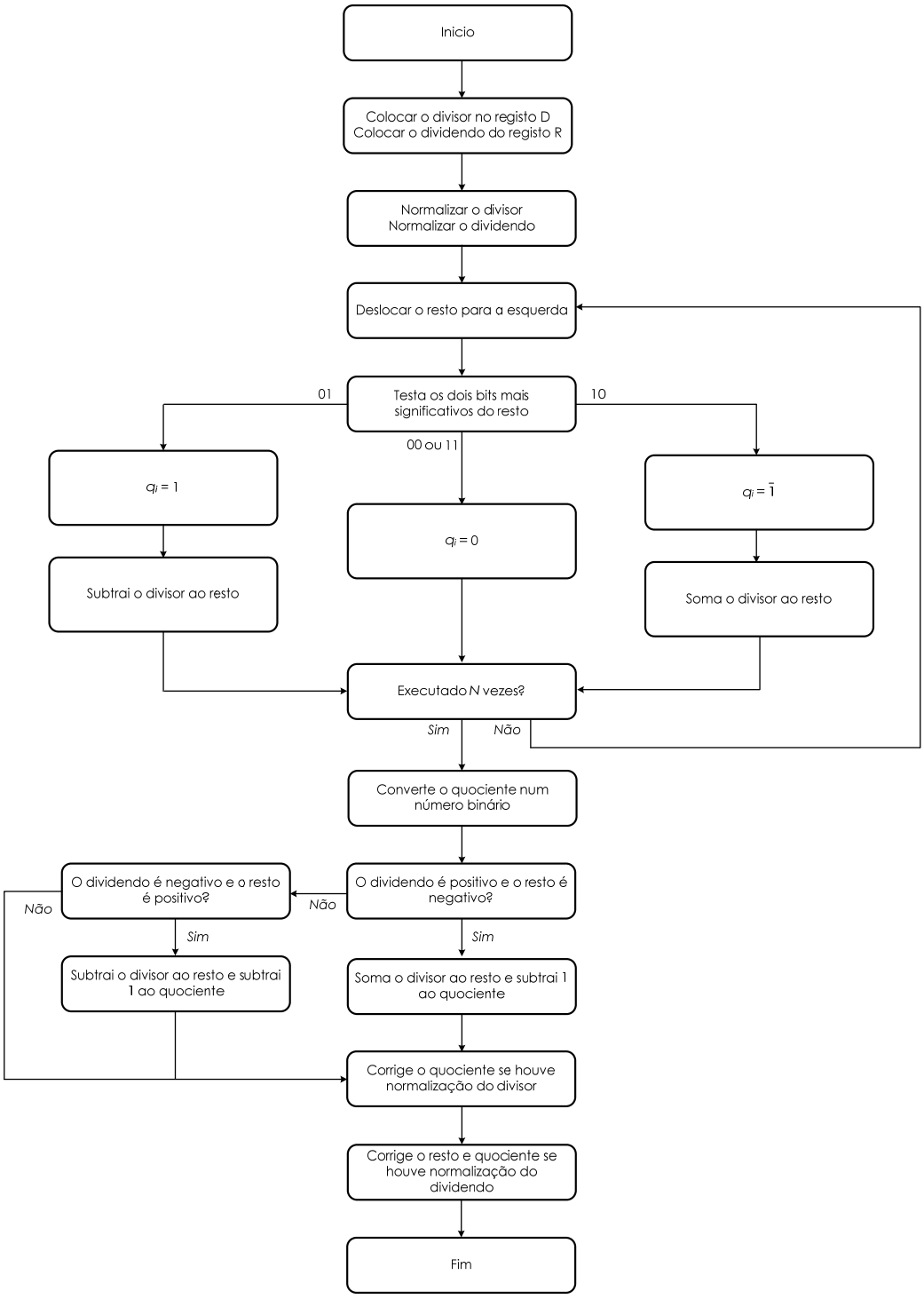


Figura 3.81 – Fluxograma do divisor SRT.

O procedimento acaba quando o resto for menor do que o divisor. Ao realizar-se a divisão de dois números positivos tanto o quociente como o resto devem ser positivos. No caso do divisor SRT, devido à possibilidade de estimativas erradas dos bits do quociente, pode ser necessário no fim corrigir o resto somando a este o divisor. Nesse caso é também necessário subtrair 1 ao quociente. Por exemplo, imagine-se que a divisão de 42 por 5 resultou num quociente de 8 e um resto de -3,

$$42 = 9 \times 5 - 3 \,. \tag{20}$$

O resto corrigido obtém-se somado ao valor negativo obtido (-3) o valor do divisor (5)dando origem a assim a um resto de 2. Para além disso o quociente passa de 9 para 8 (subtracção de 1) de modo a ter-se

$$42 = 8 \times 5 + 2 \,. \tag{20}$$

Quando o dividendo e/ou o divisor são negativos há uma complicação adicional. Deve o resto ser positivo ou negativo? Por convenção o resto deve ter o mesmo sinal do dividendo (devendo ser, em módulo, menor do que o divisor). Para que isso aconteça, no fim da divisão pode se necessário somar ou subtrair ao resto o valor do divisor e ao mesmo tempo alterar o quociente somando ou subtraindo 1. Se o dividendo for negativo e o resto positivo, como em

$$-42 = 9 \times (-5) + 3 \,, \tag{20}$$

há que subtrair 5 ai resto e subtrair 1 ao quociente. Tem-se assim

$$-42 = 8 \times (-5) - 2 \,. \tag{20}$$

O quociente será positivo ou negativo consoante os sinais do divisor e dividendo são iguais ou diferentes respectivamente.

Há também que converter o quociente de uma forma redundante para o formato binário em complemento para 2. Uma das formas de fazer isso consiste em preencher um registo (Q_P) com os valores 0 e 1 consoante os bits estimados do quociente sejam 0 ou 1 e preencher outro registo (Q_N) com 0 ou 1 consoante os bits estimados do quociente sejam 0 ou $\bar{1}$. O quociente em formato binário é então obtido subtraindo Q_N de Q_P.

$$Q_P - Q_N \rightarrow Q \,. \tag{20}$$

Há no entanto outra forma que permite determinar o valor do quociente sem ser necessário realizar a subtracção final [8][9]. Consiste em ter-se dois registos, Q e Q_M, e em cada iteração deslocá-los para a esquerda, ajustar o valor do bit menos significativo e eventualmente carregar o conteúdo de um deles no outro, consoante o valor do bit estimado para o quociente:

- q_i = 1 - desloca Q para a esquerda, copia Q para Q_M e coloca 1 no LSB de Q.
- q_i = 0 - desloca Q e Q_M para a esquerda. Coloca 1 no LSB de Q_M.
- $q_i = \bar{1}$ - desloca Q_M para a esquerda, copia Q_M para Q e coloca 1 no LSB de Q.

Veja-se agora um exemplo da divisão de dois números.

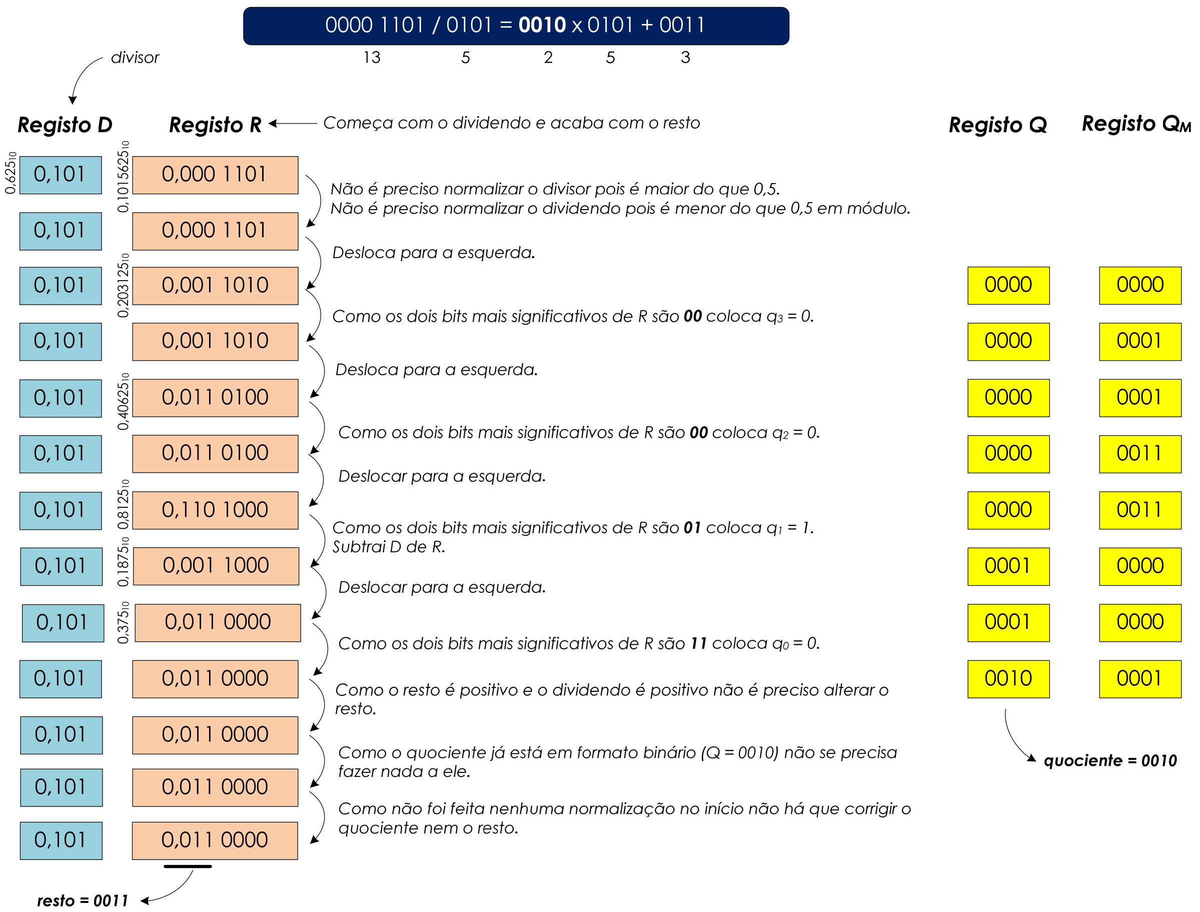


Figura 3.82 – Ilustração da evolução dos registos num divisor SRT (exemplo para 13/5).

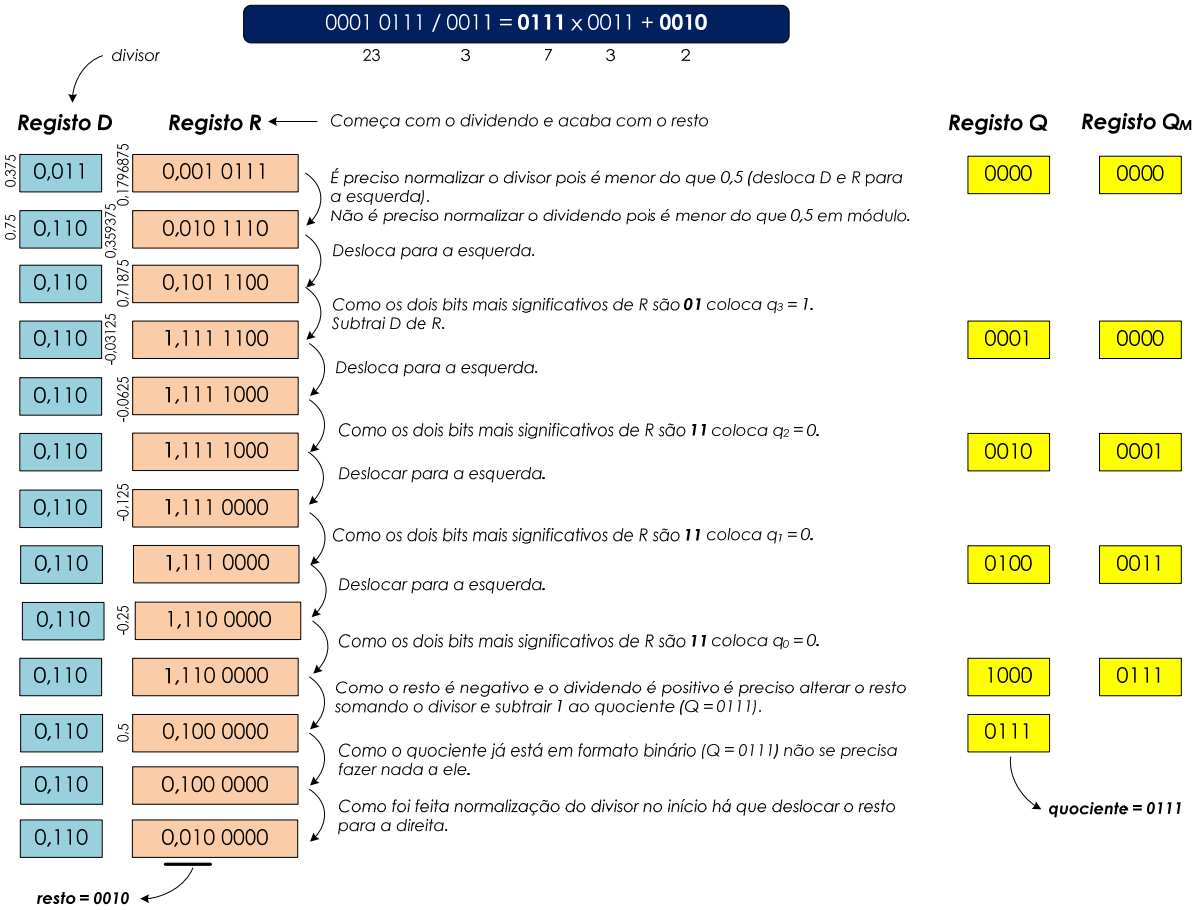


Figura 3.83 – Ilustração da evolução dos registos num divisor SRT (exemplo para 23/3).

A Figura 3.84 mostra a implementação de um divisor SRT.

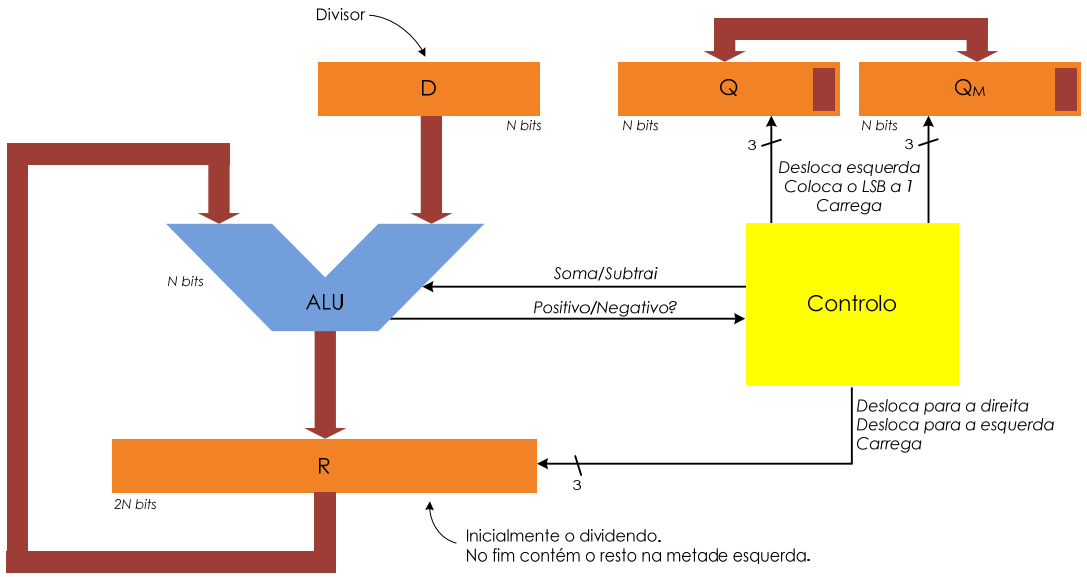


Figura 3.84 – Diagrama de blocos de um divisor SRT.

O algoritmo utilizado no divisor SRT tem em comum com o multiplicador do tipo Booth o facto de sequências de 0s ou 1s no dividendo traduzem-se por operações de soma ou subtracção só no inicio e fim das sequências. Relembre-se que quando os dois bits mais significativos do resto parcial forem 00 ou 11 a estimativa do bit de quociente é 0 e nada é somado ou subtraído ao resto parcial.

A descrição apresentada do divisor SRT teve como objectivo introduzir unicamente as ideias usadas neste tipo de divisor. Existem, no entanto, diversas alternativas que são exploradas para tornar a implementação mais eficiente em termos de área ocupada e rapidez de execução. Um dos aspectos melhorados consiste no uso de um maior número de diferentes estimativas para os bits de quociente. Em vez de se usarem só 3 como aqui descrito (-1, 0 e 1) existem alternativas onde são usados 5 (-2, -1, 0, 1 e 2) ou mais. O caso aqui apresentado é designado de divisor SRT de base-2 (*radix*-2, em inglês) enquanto que aquele onde são usadas 5 alternativas é designado por divisor SRT de base-4 (*radix*-4, em inglês).

Outra melhoria consiste em usar somadores/subtractores do tipo carry-save para somar/subtrair o divisor do resto parcial. A vantagem é que durante a determinação de cada bit de quociente as somas e subtracções são efectuadas sem propagação de carry o que permite fazer a soma e subtracção de todos os bits em paralelo. Relembre-se que o resultado da soma neste casa são dois bits: o bit de soma e o bit de carry. Só no fim do procedimento, quando todos os bits do quociente foram determinados, é que é feita a propagação do carry obtendo-se o valor do resto da divisão. Neste caso tanto a ALU como o registo R têm de estar preparados para usar os restos parciais representados nessa forma. Também o bloco de controlo fica mais complicado pois deve decidir os bits de quociente com base nos bits mais significativos do resto parcial representado nesta forma. A poupança de tempo de execução é tanto maior quanto mais bits forem usados para representar os números. Em casos de processadores de 32 ou 64 bits o ganho pelo uso dos somadores/subtractores carry-save é tanto que até deixa de se justificar o uso do algoritmo aqui descrito quando comparado com os outros divisores mais simples pois o tempo que demora somar e subtrair dois números é da mesma ordem de grandeza do que o tempo da lógica necessária para seleccionar o bit de quociente correcto.

3.6 Circuitos de Deslocamento

Os circuitos de deslocamento possibilitam deslocar uma palavra binária para a esquerda ou para a direita de um dado número de posições. Existem diversos tipos de circuito consoante o que acontece com o valor do bit correspondente à posição que ficou livre no caso do deslocamento de uma posição (o bit mais significativo ou o bit menos significativo):

- Circuito de deslocamento lógico: a posição vaga á preenchida com 0.
- Circuito de deslocamento aritmético: a posição vaga mantém o valor anterior.
- Circuito de deslocamento rotativo: a posição vaga á preenchida com o bit que sai do registo no outro extremo.

De seguida apresentam-se dois circuitos que possibilitam o deslocamento de uma palavra: usando multiplexers e usando uma matriz de transístores.

3.6.1 Utilizando Multiplexers

A Figura 3.85 apresenta o diagrama de blocos de um circuito constituído por multiplexers que possibilita o deslocamento do conteúdo de um registo de um número de posições definido pelos bits s_0, s_1 e s_2. Por exemplo, caso esses bits tenham o valor 110 a saída b_0 vai conter o valor da entrada a_3. Há assim um deslocamento de 3 posições para a direita da palavra a.

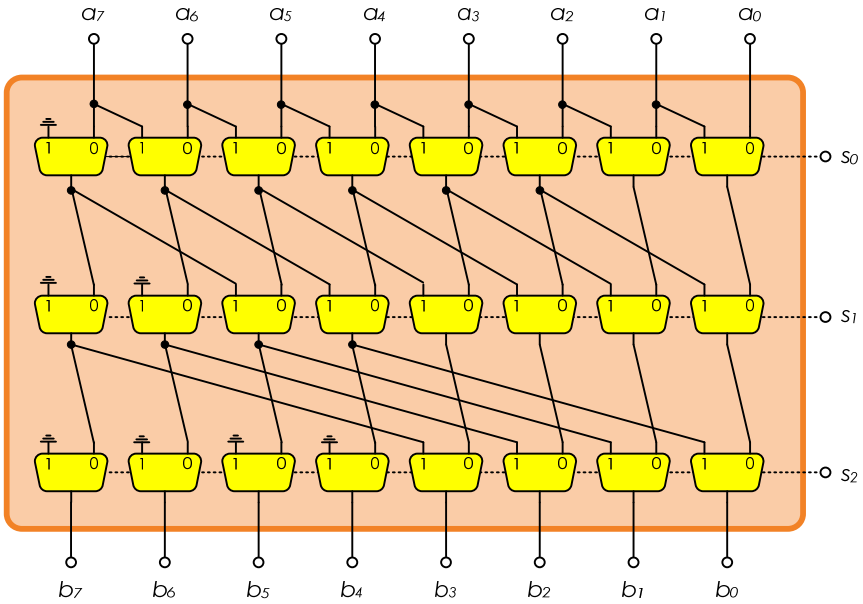


Figura 3.85 – Diagrama de blocos de parte de um registo de deslocamento constituído por multiplexers.

3.6.2 Utilizando uma Matriz de Transístores

A Figura 3.86 mostra parte do circuito de um registo de deslocamento aritmético constituído por uma matriz de transístores (*barrel shifter*). As linhas s_0, s_1, s_2 e s_3 seleccionam quais os transístores que estão a conduzir. Se só a linha S_1 estiver com o nível alto, por exemplo, a saída b_0 ficará ligada à entrada a_1, a saída b_1 à entrada a_2 e assim sucessivamente. Dá-se assim um deslocamento para a direita de uma posição.

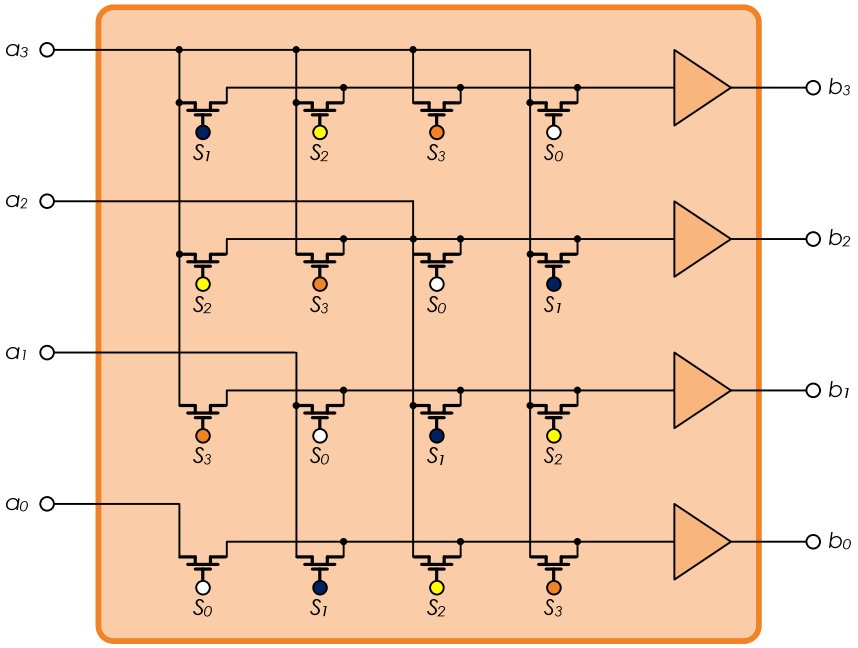


Figura 3.86 – Diagrama de blocos de parte de um registo de deslocamento aritmético para a esquerda constituído por uma matriz de transístores.

A Figura 3.87 mostra um circuito de deslocamento lógico em vez de aritmético.

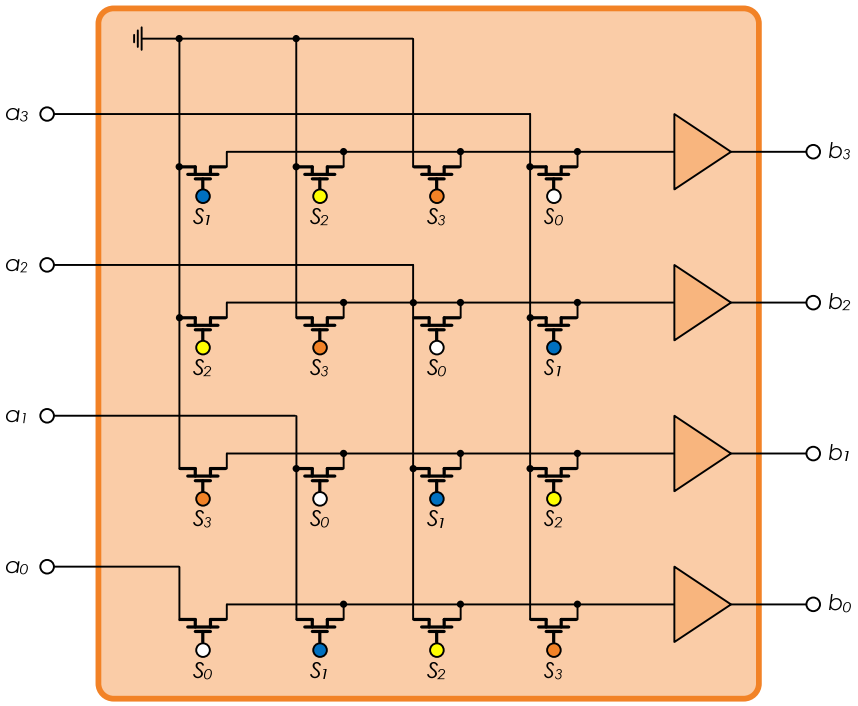


Figura 3.87 – Diagrama de blocos de parte de um registo de deslocamento lógico para a direita constituído por uma matriz de transístores.

Note-se que a maior parte do espaço ocupado por este circuito num circuito integrado é devido às ligações e não aos transístores utilizados.

3.7 Representação em Vírgula Flutuante

3.7.1 Notação Científica

Muitas aplicações necessitam de trabalhar com números que não são inteiros, isto é, com números reais. Tendo em conta que um computador tem uma capacidade de memória limitada, não é possível representar um número arbitrário qualquer como por exemplo π já que contém um número um número infinito de dígitos. É, no entanto, possível representar um grande conjunto de números reais.

Isso é conseguido através do uso de uma representação que é comum designar por ***representação em vírgula flutuante***. Essa representação é semelhante à notação científica e o seu nome realça a diferença para um outro tipo de representação de números reais designada por ***representação em vírgula fixa***. A principal vantagem da representação em vírgula flutuante é a muito maior gama de valores que é capaz de representar.

A notação científica é uma forma de representação que tenta por em evidência a ordem de grandeza de um número. Isso é feito separando a representação de um dado número (x) em outros dois números designados por mantissa (*m*) e expoente (*e*). A Figura 3.88 ilustra como é feita essa separação. O número é representado pelo produto da mantissa por uma potência da base (*b*) em que se esta a trabalhar.

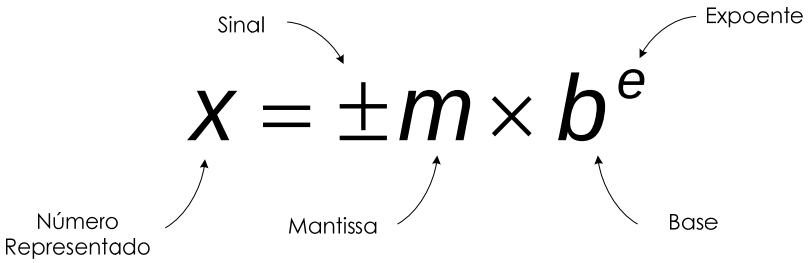


Figura 3.88 – Notação científica para usada representar um número.

No caso da base 10, por exemplo, um expoente de 2 significa que o número representado é 100 vezes (10^2) maior do que a mantissa.

Note-se que além da mantissa, expoente e base, que geralmente está subentendida, é necessária ainda informação de sinal (*s*) para indicar se o número é positivo ou negativo.

O número real 1526,889, por exemplo, pode ser representado em notação científica usando $1{,}526889 \times 10^3$. A mantissa é 1,52889, o expoente é 3 e a base é 10. Essa notação não é, no entanto, única. A representação $15{,}26889 \times 10^2$ é outra possibilidade em que a mantissa é dez vezes maior e o expoente é uma unidade inferior do que no caso anterior. A primeira versão chama-se de representação ***normalizada*** por só conter um algarismo à esquerda da vírgula.

A representação em notação científica tem duas vantagens. Por um lado transmite facilmente uma indicação sobre a ordem de grandeza do número e por outro lado põe em evidência os algarismos

significativos. A massa do electrão, por exemplo, pode ser expressa usando $9{,}1093822 \times 10^{-31}$ kg, em vez de ser expressa usando 0,000000000000000000000000000091093822 kg. O expoente "-31" dá imediatamente uma ideia sobre a ordem de grandeza do número.

3.7.2 Representação em Vírgula Flutuante Segundo a Norma IEEE 754

Existem diferentes formas de implementar o uso da notação científica num computador. A adoptada normalmente é a definida na norma IEEE 754. Essa norma especifica diferentes formatos com precisões diferentes (Tabela 3.10).

Tabela 3.10 – Formatos de representação de números em vírgula flutuante previstos na norma IEEE 754-2008.

Type	Sign	Exponent	Significand	Total bits	Exponent bias	Bits precision
Half (IEEE 754-2008)	1	5	10	16	15	11
Single	1	8	23	32	127	24
Double	1	11	52	64	1023	53
Quad	1	15	112	128	16383	113

O formato "Single", por exemplo, usa 32 bits para representar os números. O bit mais significativo é usado para o sinal, os 8 seguintes são usados para o expoente e os últimos 23 são usados para a mantissa que deve estar normalizada (Figura 3.89).

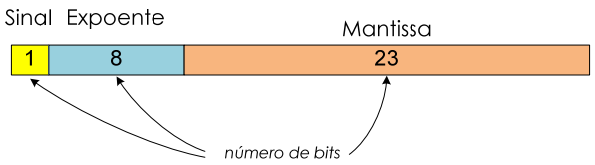


Figura 3.89 – Distribuição dos 32 bits de uma representação em vírgula flutuante de acordo com a norma IEEE 754.

No caso de representação binária, o "1" à esquerda da vírgula não é representado explicitamente (Figura 3.90). Note-se que no caso decimal isso não acontece pois existem 9 valor possíveis para o algarismo mais à esquerda.

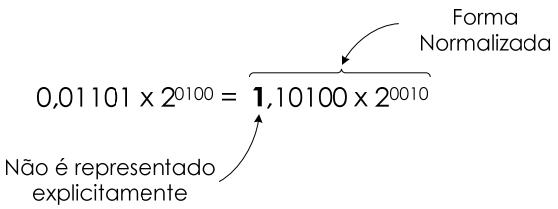


Figura 3.90 – Exemplo da normalização de um número binário em vírgula flutuante.

Os 8 bits usados para representar o expoente contêm, na verdade, o valor do expoente somado de 127. Isso permite que esse valor seja sempre positivo o que torna a comparação entre números mais fácil. A gama de valores possíveis para o expoente da notação científica é de -126 a 127. O valor que

é de facto armazenado no campo do exponente (8 bits) da sua representação em vírgula flutuante é um número entre 1 e 254.

A norma IEEE 754 prevê a representação de diversos casos especiais indicados na Tabela 3.11. O número 0, por exemplo, é representado com o valor 0 na mantissa e no expoente. Números que são demasiado pequenos para serem normalizados são representados com um 0 no campo do expoente. Um valor de 255 no campo do expoente permite representar o valor infinito ou que não se trata de um número (NaN – *Not a number*), consoante a mantissa.

Tabela 3.11 – Casos especiais previstos na norma IEEE 754.

Valor do Expoente na Representação com Vírgula Flutuante	Mantissa	Valor
0	0	± 0
0	m (≠ 0)	0, $m \times 2^{-126}$
1 a 254	m	1, $m \times 2^{e}$
255	0	± ∞
255	m (≠ 0)	NaN (Not a Number)

3.7.3 Operações Aritméticas

A multiplicação de dois números em notação cientifica realiza-se multiplicando as mantissas e somando os expoentes. A divisão, por seu lado, consiste na divisão das mantissas e subtracção dos expoentes.

No caso da adição e subtracção o procedimento é mais complexo pois é necessário ter dois números com o mesmo expoente de forma a somar ou subtrair as mantissas.

4. PROCESSADORES

4.1 Processadores Dedicados

Cada vez mais encontramos dispositivos que são capazes de realizar inúmeras funções tornando-se mais parecidos com computadores de secretária. Um telefone, que antes limitava-se a estabelecer uma chamada telefónica baseada nas teclas que eram carregadas e enviar/receber os sinais de voz, hoje em dia é capaz de fazer muito mais, desde guardar uma lista com o nome e número de telefone de várias pessoas como tirar fotografias e aceder à internet. Uma televisão que limitava-se a sintonizar os canais e mudar o volume, saturação e brilho da imagem, hoje é capaz de associar um nome a cada canal, mostrar texto envidado junto com o sinal de imagem (teletexto), mudar o tamanho da imagem no ecrã e aplicar diversos filtros à imagem.

Existem, no entanto, diversas aplicações em que os cálculos e operações a realizar são sempre os mesmos e pela mesma ordem como por exemplo um controlador para um visor LCD. A única coisa que tem que fazer é receber os dados e enviar os sinais certos para o visor de modo a acender os segmentos luminosos apropriados. Outro exemplo é o de um controlador de motor passo-a-passo. A única coisa que precisa fazer é receber os sinais de comando que indicam a direcção e ritmo do movimento e aplicar as correntes adequadas aos enrolamentos do estator do motor passo-a-passo. Nenhuma destas aplicações necessita de um processador de texto ou de ser capaz de tirar fotografias.

Nestas aplicações utilizam-se Processadores Dedicados. Este tipo de processador é projectado com um objectivo muito específico e por essa razão a sua arquitectura é algo diferente da dos processadores de uso geral. Há processadores dedicados que são genéricos na medida em que podem executar uma função bem definida que pode ser útil em diversas aplicações. Alguns exemplos são os temporizadores, interfaces de entrada/saída e relógios de tempo real. Por outro lado, há também processadores dedicados que são costumizados, isto é, desenvolvidos com um produto especifico em vista. Processadores para uso num telefone fixo, ou em brinquedos electrónicos ou para controlar um airbag num automóvel, são exemplos de processadores dedicados costumizados.

O que leva à escolha do tipo de processador a usar é fundamentalmente o custo, sendo também importante a aplicação, o tamanho, o consumo energético e o desempenho. Um processador dedicado genérico terá um custo baixo pois são produzidas muitas unidades o que amortiza o custo do desenvolvimento. Nesses casos o fabricante de um dado produto não precisa de ele próprio realizar o desenvolvimento do processador. Esse é o principal motivo que leva a que mesmo em aplicações que seriam adequado o uso de um processador dedicado costumizado, como em brinquedos electrónicos, os fabricantes acabem por escolher processadores de uso geral como microcontroladores. Evitam assim ter que desenvolver um processador dedicado. O custo do próprio processador é muito mais baixo. No caso de microcontroladores, por exemplo, o número de unidades produzidas é tão grande que o seu custo é praticamente irrelevante sendo possível encontrar microcontroladores que custam algumas dezenas de cêntimos.

O desenvolvimento de um processador dedicado costumizado tem no entanto a vantagem de permitir optimizar o desempenho para a aplicação específica a que se destina. Não só as tarefas executadas podem ser realizadas em menos ciclos de relógio, também a própria frequência de relógio pode ser maior. A especificidade das tarefas leva que o circuito seja mais simples já que as diferentes unidades de cálculo internas podem transmitir os dados entre si não sendo necessário o uso de registos intermédios ou barramentos complicados. Por outro lado, o próprio controlo da execução é mais simples, não sendo necessária memória para armazenar o programa nem a lógica para implementar instruções desnecessárias. Tudo isso faz com que o tamanho do circuito seja menor o que pode ser explorado, por exemplo, para realização de mais operações em paralelo o que aumenta ainda mais a rapidez de execução.

Os processadores dedicados muitas vezes são usados ao lado de processadores de uso geral para implementar de forma óptima algumas funções como a comunicação com periféricos ou o armazenamento de dados em memória. Existem mesmo caso de processadores de uso geral, como microcontroladores, que integram esses processadores dedicados dentro do mesmo encapsulamento ou circuito integrado. Não deixam no entanto de ser processadores dedicados com tarefas bem definidas.

De seguida apresenta-se a constituição de um processador dedicado e dão-se alguns exemplos concretos como forma de ilustrar o que está em jogo no projecto de um desses processadores.

4.1.1 Arquitectura

Devido à complexidade da constituição e funcionamento interno de um processador é comum dividi-lo em blocos organizados de forma hierárquica desde o nível mais alto, onde é constituído por um bloco onde entram dados e instruções (no caso de processadores de uso geral) e de onde saem novos dados (resultados do processamento) (Figura 4.1), até ao nível mais baixo onde o processador é descrito em termos de transístores, resistências e outros componentes electrónicos fundamentais.

Figura 4.1 – Descrição de um processador como um dispositivo de processamento de informação.

Num segundo nível da hierarquia é comum representar um processador como sendo constituído por dois blocos: a Unidade de Processamento e a Unidade de Controlo (Figura 4.2). A Unidade de Processamento (*datapath*) é onde é realizado o processamento dos dados, isto é, onde são efectuadas as computações propriamente ditas. A Unidade de Controlo, por seu lado, é responsável pelo controlo da Unidade de Processamento tanto a nível do fluxo de dados através dessa unidade como através da selecção da operação a realizar em cada instante.

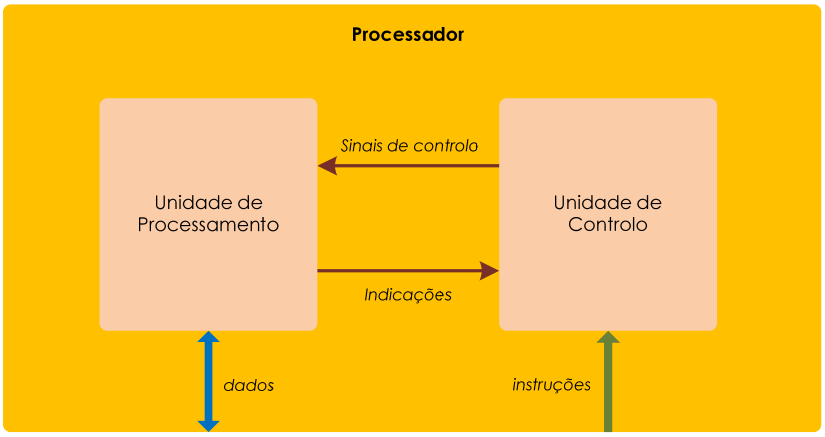


Figura 4.2 – Descrição de um processador como sendo constituído por uma Unidade de Processamento e por uma Unidade de Controlo.

Esta descrição é necessariamente simplificada havendo lugar a variações de processador para processador. A Unidade de Controlo, por exemplo, pode receber sinais de controlo directamente do exterior assim como pode enviar sinais para o exterior com informação sobre o estado do processador. No caso de processadores de uso geral a Unidade de Controlo recebe do exterior as instruções com as operações a realizar em cada instante enquanto nos processadores dedicados a Unidade de Controlo determina, só por si, a sequência de operações a realizar.

Num terceiro nível hierárquico pode-se descrever a unidade de processamento como sendo constituída pela Unidade Aritmética e Lógica, onde são realizadas as operações matemáticas (soma subtracção, 'e' lógico, 'ou' lógico, deslocamento de bit, etc.) e o Ficheiro de Registos, constituído por vários registos onde são armazenados temporariamente os operandos para as operações matemáticas e os resultados dessas operações (Figura 4.3).

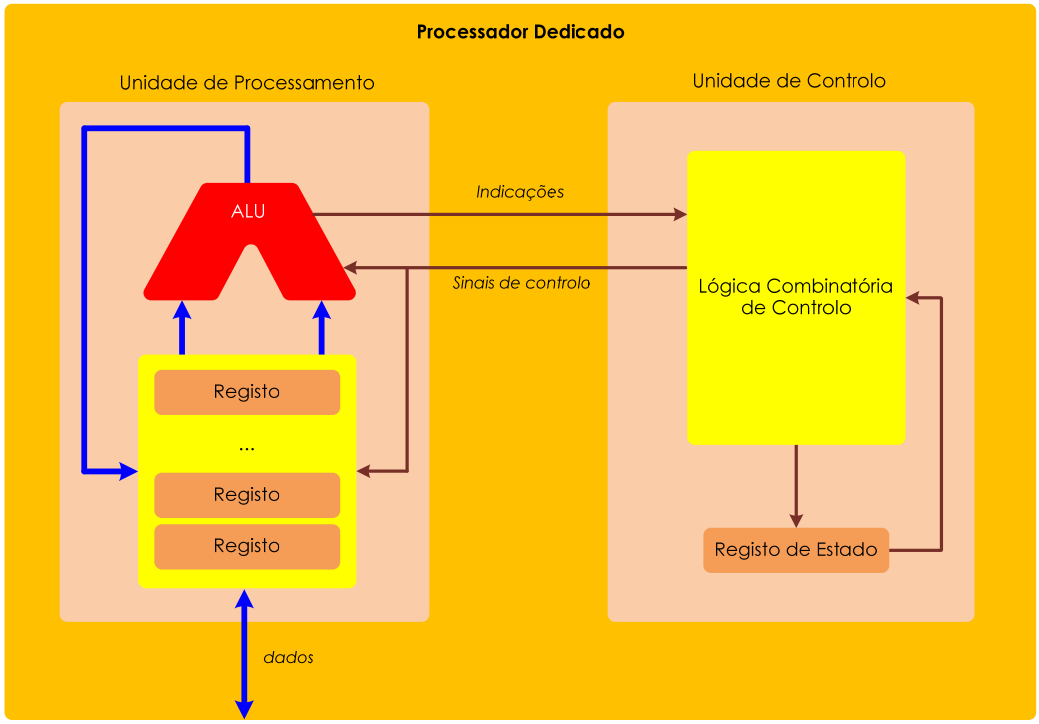


Figura 4.3 – Constituição da Unidade de Processamento e da Unidade de Controlo num processador dedicado.

Nos processadores dedicados a Unidade de Controlo é realizada através da implementação de uma máquina de estados finita. Essa implementação consiste num registo para armazenamento do estado actual da máquina de estados e de um circuito lógico responsável por determinar os sinais de saída e o próximo estado com base nos sinais de entrada e no valor do estado actual.

4.1.2 Projecto

O projecto de um processador dedicado consiste no desenvolvimento da Unidade de Processamento e da Unidade de Controlo.

O início do projecto consiste em descrever o algoritmo usando pseudo-código ou um fluxograma. Esse algoritmo é composto por operações de cálculo, atribuição de variáveis, ciclos de operações (*while*) e por saltos condicionais (*if*).

A forma comum de transpor o algoritmo desejado para uma implementação em hardware usando circuitos digitais lógicos consiste em descrever a operação do processador através de uma máquina de estado finita. Essa máquina de estados finita descreve não só a unidade de controlo mas também a unidade de processamento. Por essa razão é costume designar-se está descrição por "máquina de estados finita com circuito de dados" (FSMD – *finite state machine with datapath*).

Para se criar a FSMD há que associar um estado a cada passo do algoritmo. A criação desses estados para atribuições, ciclos e saltos condicionais é ilustrada na Figura 4.4.

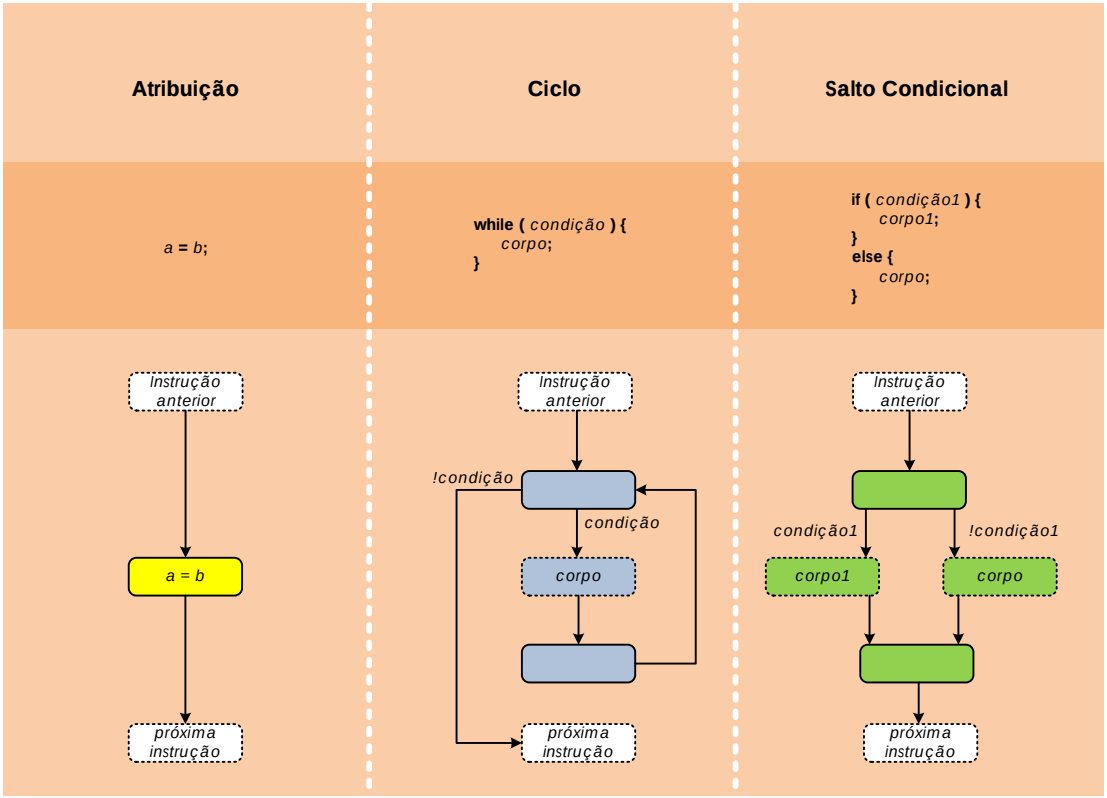


Figura 4.4 – Realização em máquina de estados finita de diferentes elementos de um algoritmo.

Há então que construir a unidade de processamento onde se colocam os diferentes elementos da unidade aritmética e lógica consoante as funções matemáticas contidas no algoritmo. É preciso também incluir um registo para cada variável utilizada e multiplexers quando a uma mesma variável são atribuídas diferentes valores em diferentes partes do algoritmo (diferentes estados da FSMD). Cada registo precisa de um sinal de controlo que determina quando a palavra à sua entrada é armazenada. Os multiplexers precisam de um ou mais sinais para determinar qual das suas entradas é redireccionada para a saída.

Quanto à unidade de controlo é necessário criar a sua máquina de estados o que é feito a partir da FSMD do processador substituindo as atribuições de variáveis e operações matemáticas aí existentes por alterações ou verificações de valores de sinais de controlo necessários pela unidade de processamento e com base nos sinais de indicação fornecidos por esta.

Finalmente há que implementar a lógica necessária para determinar as transições de estado escrevendo as expressões lógicas dos sinais de controlo e simplificando-as por intermédio, por exemplo, de mapas de Karnaugh. Essas expressões lógicas podem então ser implementadas através de portas lógicas. O tamanho do registo de estado da unidade de controlo depende do número de estados que a máquina de estados do controlador tem.

4.1.3 Optimização

O projecto de um processador dedicado para uma determinada função não tem uma única solução. Há sempre que ter em conta diferentes métricas, como o custo, a área ocupada, o consumo energético e a rapidez de execução.

A criação da FSMD através da transposição dos diferentes elementos do algoritmo usando os elementos apresentados na Figura 4.4 não leva necessariamente à máquina de estados mais compacta possível. É desejável que, a posteriori, ela seja optimizada através da junção de estados diferentes ou à eliminação de estados desnecessários. Quando o algoritmo contém duas atribuições consecutivas de variáveis diferentes, por exemplo, é possível juntar os dois estados resultantes num só já que essas operações podem ser executadas em paralelo. Relembre-se que o uso de uma máquina de estados para implementar um circuito lógico tem implícita a criação de um algoritmo sequencial. Como em qualquer algoritmo existem partes que são independentes existe toda a vantagem em executá-las em paralelo.

Outra área onde é possível realizar a optimização é ao nível do algoritmo. Existindo diferentes formas de realizar uma certa função há que escolher o algoritmo que mais se adequa às necessidades da aplicação e do projectista. O cálculo de x^y (com x e y inteiros) pode ser feito, por exemplo, usando um acumulador inicializado com o valor de x e que é, em cada uma das y - 1 iterações de um ciclo, multiplicado por x. Este algoritmo tem a vantagem de ser conceptualmente simples e de usar unidades aritméticas relativamente simples como a multiplicação mas tem a desvantagem de ser lento e do tempo de cálculo aumentar com o valor de y. Outra solução seria, por exemplo, calcular o logaritmo de x, multiplicar o resultado por y e finalmente calcular a potência de 10 do resultado:

$$x^y = 10^{y \log(x)}. \tag{21}$$

Esta solução tem a vantagem de ter um tempo de cálculo que não depende de y mas necessitar de uma unidade aritmética complexa para o cálculo do logaritmo e da potência de 10. Para além disso, essas unidades aritméticas teriam que operar com números em vírgula flutuante.

A constituição da unidade funcional pode também ser optimizada. Pode-se tentar, por exemplo, usar o menor número de variáveis possível com o objectivo de diminuir o número de registos presentes na unidade funcional (*datapath*). No caso da realização das operações matemáticas pode-se optar por usar o menor número de unidades aritméticas ou lógicas reutilizando-as para diferentes funções (realizar o simétrico de um número com um subtractor que faz a diferença entre 0 e o número em causa, por exemplo) ou, pelo contrário, ter uma unidade dessas para cada operação matemática necessária. A primeira solução permite ter um menor consumo e uma menor área total do processador e consequentemente um custo mais reduzido enquanto a segunda solução torna o projecto do processador menos complexo e permite que diferentes operações sejam realizadas em simultâneo levando a uma maior rapidez de operação.

4.1.4 Exemplo

Veja-se, como exemplo, o cálculo de x^y com x e y inteiros positivos. Um algoritmo possível consiste em multiplicar x por si próprio y – 1 vezes. A Figura 4.5 mostra o pseudo-código desse algoritmo. É usada a variável *a* para guardar o resultado dos produtos sucessivos e a variável *n* para contar o número de vezes que falta efectuar as multiplicações. Inicialmente o número de multiplicações necessário é igual a y - 1. Cada vez que se efectua uma multiplicação esse valor é decrementado de uma unidade.

Estado	Pseudo-código
	int a, n;
0	while (1) {
1	while(!ready) {
2	}
3	done = 0;
4	a = x_i;
5	n = y_i;
6	n = n – 1;
7	while(n > 0) {
8	a a * x;
9	n = n - 1;
10	}
11	z_o = a;
12	done = 1;
13	}

Figura 4.5 – Pseudo-código para implementação do cálculo de x^y.

A variável "ready" representa o sinal externo que entra no processador e que indica quando é que os dados de entrada (x e y) estão prontos de forma a dar-se início ao cálculo.

A máquina de estados finita com circuito de dados (FSMD) obtém-se do algoritmo apresentado (Figura 4.6). A cada estado foi atribuído um número diferente começando a numeração em 0 pois facilitará mais adiante a representação do estado de forma binária.

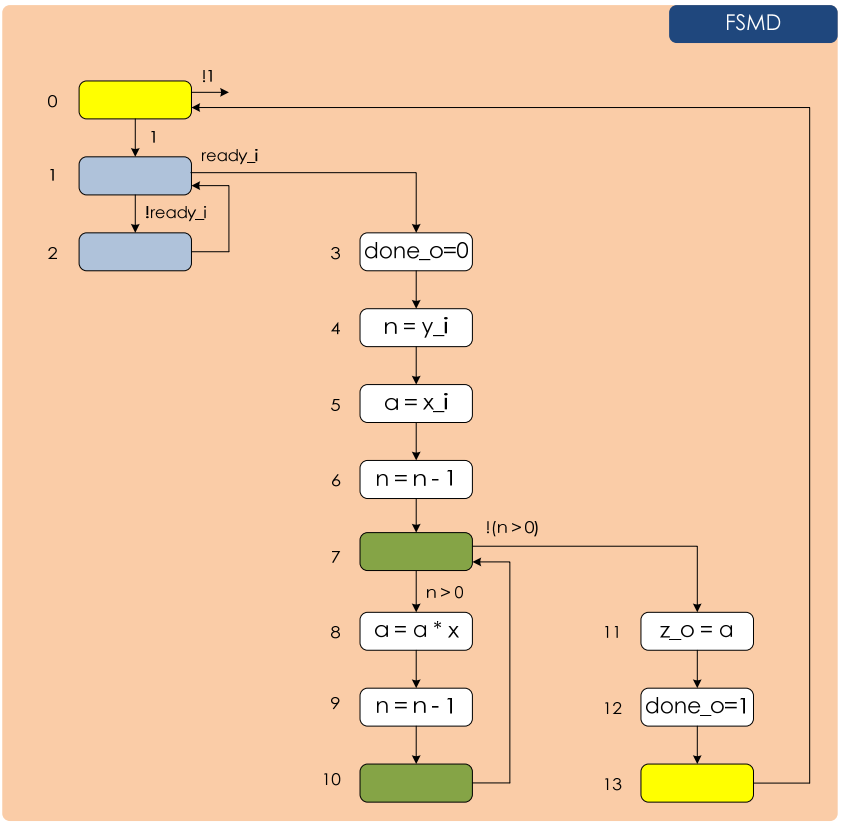


Figura 4.6 – Máquina de estados finita com ***datapath*** para um processador dedicado que determina x^y.

Esta máquina de estados pode ser optimizada juntando vários estados (3/4, 8/9 e 11/12) e eliminando os estados desnecessários que foram introduzidos no fim dos ciclos (2, 10 e 13) como se observa na Figura 4.7. Note-se que o estado 5 foi mantido (estado 2 na máquina optimizada) porque é sempre necessário haver um estado entre dois estados onde é feita uma atribuição a uma mesma variável (variável *n*, neste caso). Isso acontece porque os sinais de controlo mudam (eventualmente) de valor na entrada dos estados e porque os registos armazenam as palavras na sua entrada nos flancos dos sinais. É portanto preciso que o sinal que controla o carregamento do registo esteja no nível 0 antes de entrar no estado onde é feita a atribuição. Isso não acontece se no estado anterior houve uma atribuição à mesma variável. Neste caso, no estado 2, o sinal de controlo do registo associado à variável *n* será reposto a 0 para que no estado 4 possa haver uma nova transição para o nível 1.

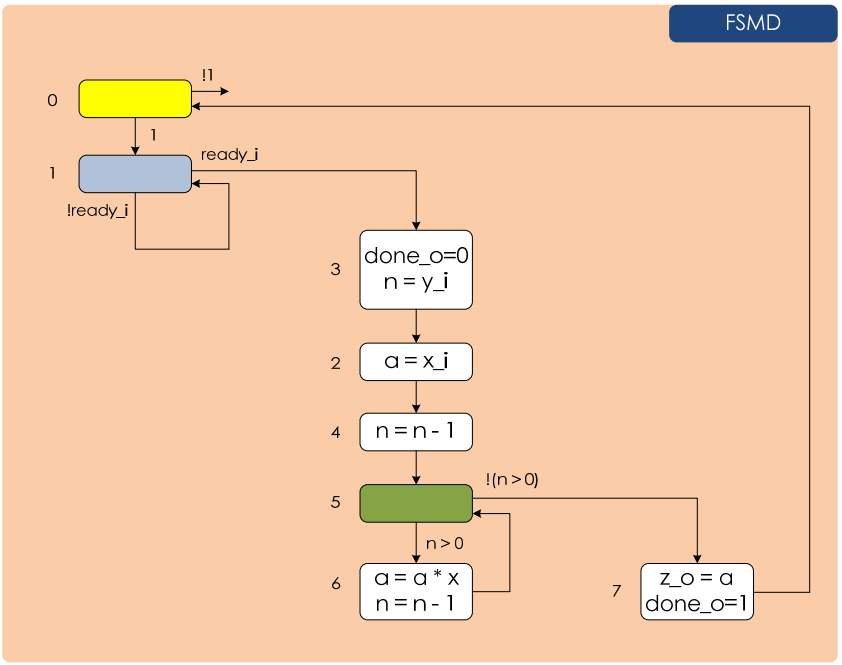


Figura 4.7 – Máquina de estados finita com ***datapath*** optimizada para um processador dedicado que determina x^y.

A escolha do número de ordem de cada estado pode ser totalmente aleatória, no entanto, há algumas escolhas que podem simplificar a determinação das funções lógicas das saídas do controlador. Uma recomendação é a de escolher números para os estados destinos de estados em que é verificada uma condição que, na sua representação binária, variem o menor número de bits e que esse ou esses bits que variam não sejam comuns com outros estados de verificação de condições que dependam de variáveis diferentes.

No exemplo em causa isso passa-se para os estados 1 e 5. No estado 1 a condição verificada diz respeito ao sinal ***ready_i***. Escolheu-se numerar os 2 estados de destino com o número 1 (001_2) e 3 (011_2) de modo a que só o 2º bit menos significativo varie. No estado 5 a condição verificada diz respeito ao sinal ***n_pos***. Escolheu-se numerar os 2 estados de destino com o número 6 (110_2) e 7 (111_2) de modo a que só o bit menos significativo varie. Se a condição verificada dissesse respeito ao sinal ***ready_i*** então devia ser o 2º bit menos significativo a variar e uma solução seria escolher os números 4 (100_2) e 6 (110_2). É esta a razão porque os estados 2 e 3 têm uma ordenação que parece pouco natural à primeira vista.

O ***datapath*** contém os registos e as unidades funcionais. Neste caso são precisos registos para as variáveis ***a*** e ***n*** e para guardar o valor final (*z*). Em relação às unidades funcionais é preciso unidades para a multiplicação, para decrementar uma unidade a um valor e para verificar se um número é maior do que zero. É necessário também um multiplexer para seleccionar o que é guardado no registo ***a*** (a variável de entrada *x_i* ou o resultado de uma multiplicação) e outro multiplexer para seleccionar o que é guardado no registo ***n*** (a variável de entrada *y_i* ou o resultado da decrementação).

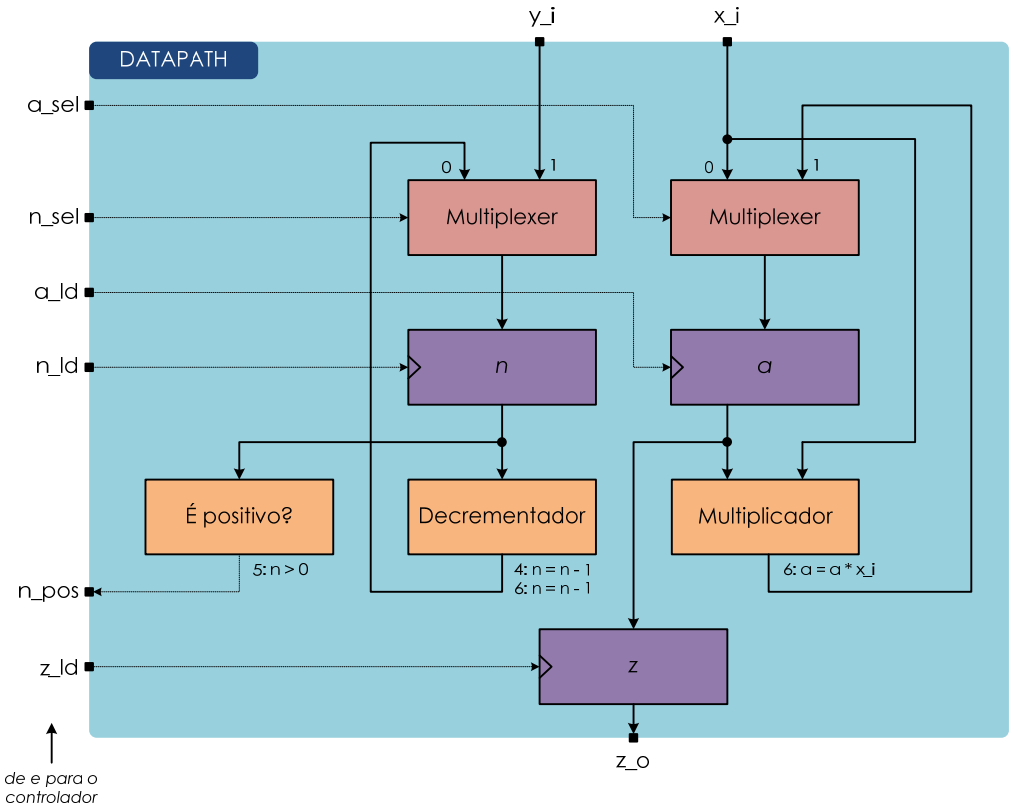


Figura 4.8 – ***Datapath*** para um processador dedicado que determina x^y.

A máquina de estados finita do controlador obtém-se da FSMD substituindo as variáveis e funções pelos sinais de controlo dos dispositivos existentes no ***datapath*** (Figura 4.9). Repare-se que o sinal n_ld é retornado a 0 nos estados 3 e 5 e que o sinal a_ld é retornado a 0 no estado 5. No estado inicial (estado 0) todas os sinais de carregamento dos registos são colocados a 0.

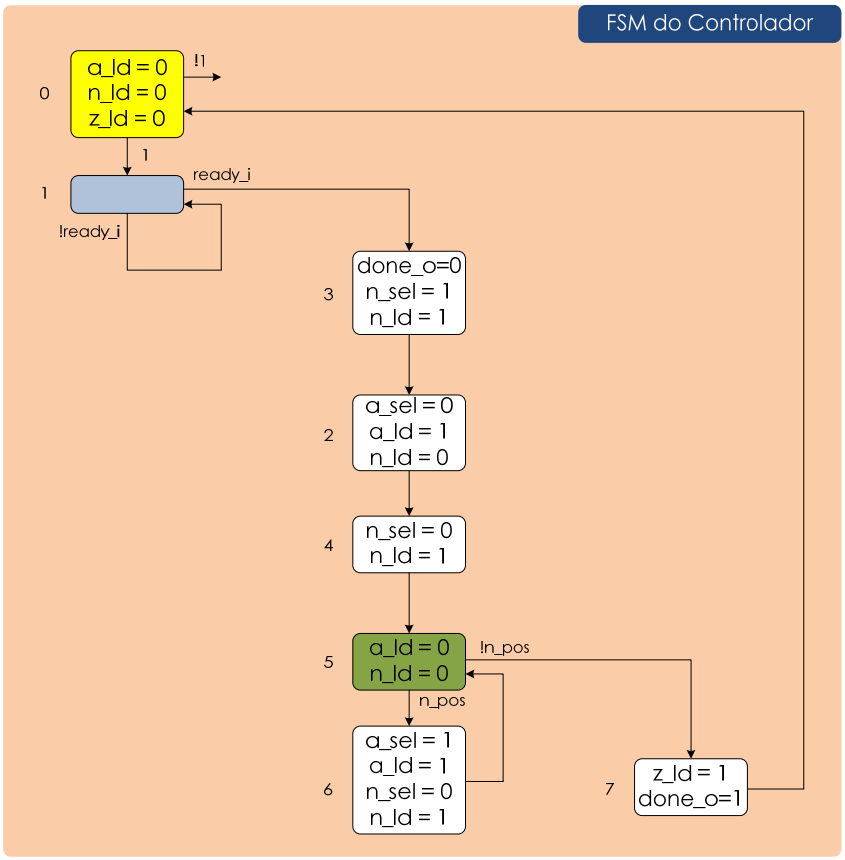


Figura 4.9 – Máquina de estados finita da unidade de controlo de um processador dedicado que determina x^y.

Neste caso obtém-se uma máquina de estados com 8 estados. O controlador precisa portanto de um registo de 3 bits para armazenar o estado actual. A Tabela 4.1 contém a tabela de verdade do controlador obtida por análise da sua máquina de estados finita. Esta tabela tem 5 colunas para as variáveis de entrada que são 3 sinais que indicam o estado actual (Q0, Q1 e Q2), um sinal externo ao processador que indica quando os dados de entrada estão prontos (*ready*) e um sinal de indicação que vem da unidade funcional e que indica se o valor da variável *n* é positivo (*n_pos*). A tabela de verdade tem 9 saídas. As 3 primeiras dizem respeito ao estado seguinte da máquina de estados (I0, I1 e I2). O sinal *done_o* serve para indicar para o exterior quando o cálculo está concluído e o resultado disponível na saída. As variáveis a e n possuem associadas 2 sinais cada uma: a_sel/n_sel, usado para controlar o multiplexer e a_ld/n_ld usados para escrever nos registos. Finalmente existe mais um sinal de controlo para o registo *z* (*z_ld*) que controla a escrita do resultado da computação.

Escolheu-se deixar em branco as células da tabela de verdade cujo valor é indiferente. Foram introduzidas 8 linhas, 1 para cada estado, tendo depois sido introduzidas uma segunda linha para os estados que dependem de variáveis de entrada. Neste caso a linha do estado 1 foi desdobrada em duas para ter em conta os dois possíveis valores da variável *ready* e a linha correspondente ao estado 5 foi desdobrada também em duas linhas para ter em conta os dois possíveis valores da variável de entrada *n_pos*.

Note-se que as colunas dos sinais de carregamento dos registos (*a_ld*, *n_ld* e *z_ld*) não têm células vazias. Nos estados em que os registos têm de ser carregados (atribuição das variáveis) esses sinais são colocado a 1 e nos restantes casos são colocados a 0.

Quanto aos sinais de selecção dos multiplexers (*a_sel* e *n_sel*) é só necessário garantir que têm o valor certo quando houver uma gravação no registo correspondente.

Tabela 4.1 – Tabela de verdade do controlador. As células em branco representam casos cujo valor é indiferente.

Estado	Entradas					Saídas								
	Q2	*Q1*	*Q0*	*ready_i*	*n_pos*	*I2*	*I1*	*I0*	*done_o*	*a_sel*	*a_ld*	*n_sel*	*n_ld*	*z_ld*
0	0	0	0			0	0	1	1		0		0	0
1	0	0	1	0		0	0	1	1		0		0	0
1	0	0	1	1		0	1	1	1		0		0	0
2	0	1	0			1	0	0	0	0	1		0	0
3	0	1	1			0	1	0	0		0	1	1	0
4	1	0	0			1	0	1	0		0	0	1	0
5	1	0	1		0	1	1	1	0		0		0	0
5	1	0	1		1	1	1	0	0		0		0	0
6	1	1	0			1	1	1	0	1	1	0	1	1
7	1	1	1			0	0	0	1		0		0	0

O passo seguinte consiste em obter as funções lógicas para cada uma das 9 saídas do controlador. É possível escrever imediatamente essa função lógica por inspecção directa da tabela de verdade no caso dos sinais *a_sel*, *n_sel* e *z_ld*,

$$\begin{aligned} a\_sel &= Q2 \\ n\_sel &= \overline{Q2} \\ z\_ld &= \overline{Q0} \cdot Q1 \cdot Q2 \end{aligned} \quad . \tag{22}$$

Para os outros sinais há que construir, por exemplo, mapas de Karnaugh para obtenção das funções lógicas (Figura 4.10).

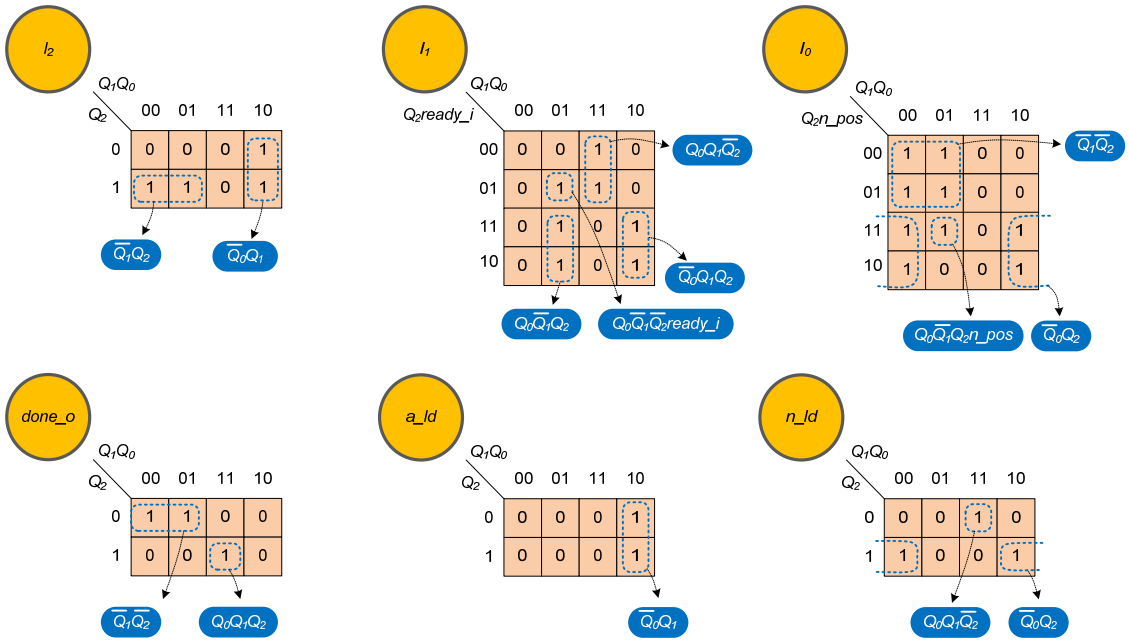


Figura 4.10 – Mapas de Karnaugh usados para a determinação das funções lógicas das saídas do controlador.

O resultado é as seguintes funções lógicas:

$$
\begin{aligned}
l_2 &= \overline{Q_0}\cdot Q_1 + \overline{Q_1}\cdot Q_2 \\
l_1 &= Q_0\cdot Q_1\cdot\overline{Q_2} + Q_0\cdot\overline{Q_1}\cdot Q_2 + \overline{Q_0}\cdot Q_1\cdot Q_2 + Q_0\cdot\overline{Q_1}\cdot\overline{Q_2}\cdot ready\_i \\
l_0 &= \overline{Q_1}\cdot\overline{Q_2} + \overline{Q_0}\cdot Q_2 + Q_0\cdot\overline{Q_1}\cdot Q_2\cdot n\_pos \\
done\_o &= \overline{Q_1}\cdot\overline{Q_2} + Q_0\cdot Q_1\cdot Q_2 \\
a\_ld &= \overline{Q_0}\cdot Q_1 \\
n\_ld &= \overline{Q_0}\cdot Q_2 + Q_0\cdot Q_1\cdot\overline{Q_2}
\end{aligned}
\quad . \qquad (23)
$$

O passo final consiste em implementar estas funções lógicas num circuito electrónico digital (Figura 4.12).

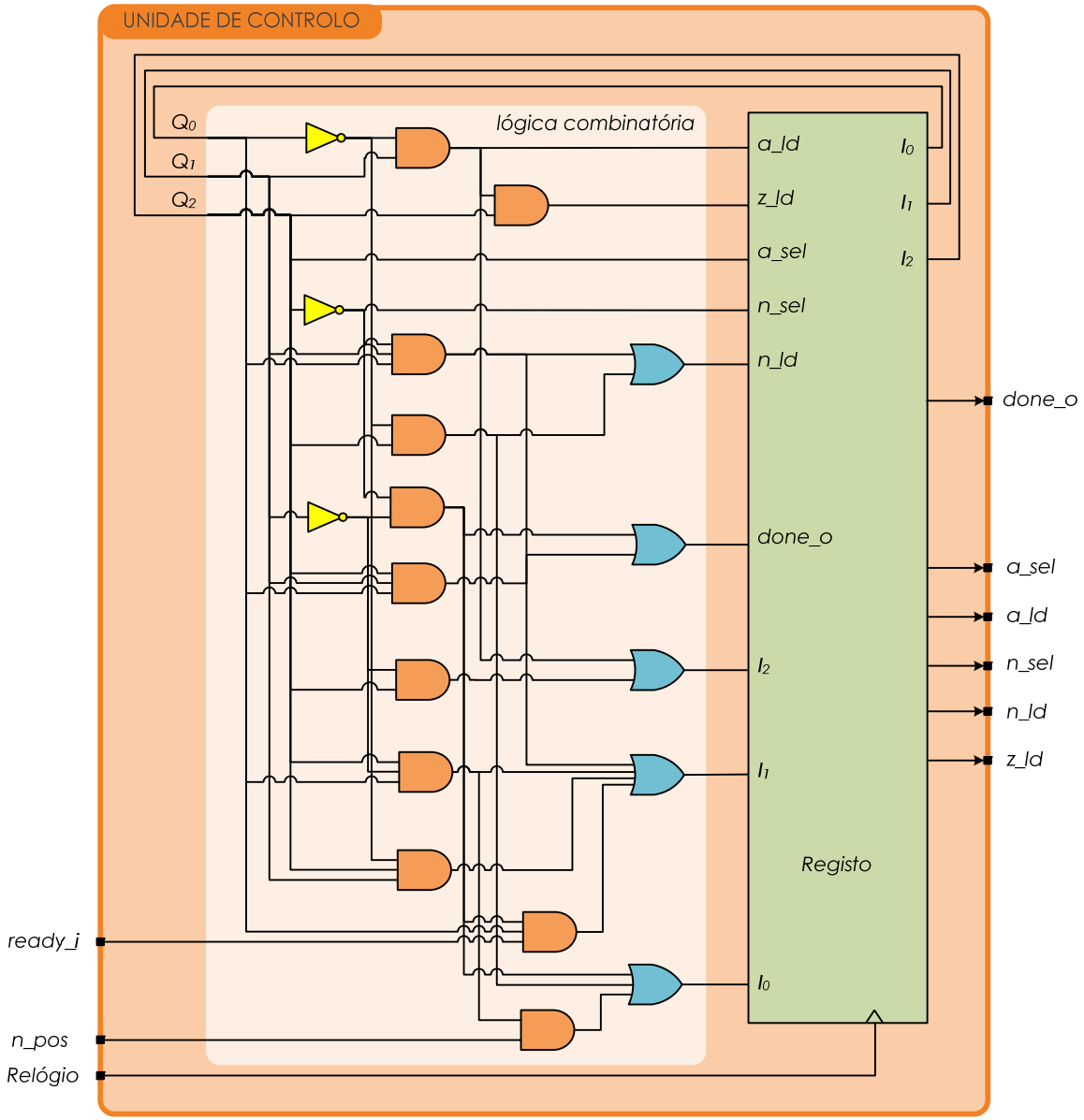


Figura 4.11 – Esquema da unidade de controlo.

Juntando a unidade de processamento (Figura 4.8) e a unidade de controlo (Figura 4.11) obtém-se o esquema eléctrico do processador dedicado que é capaz de calcular x^y (Figura 5.1).

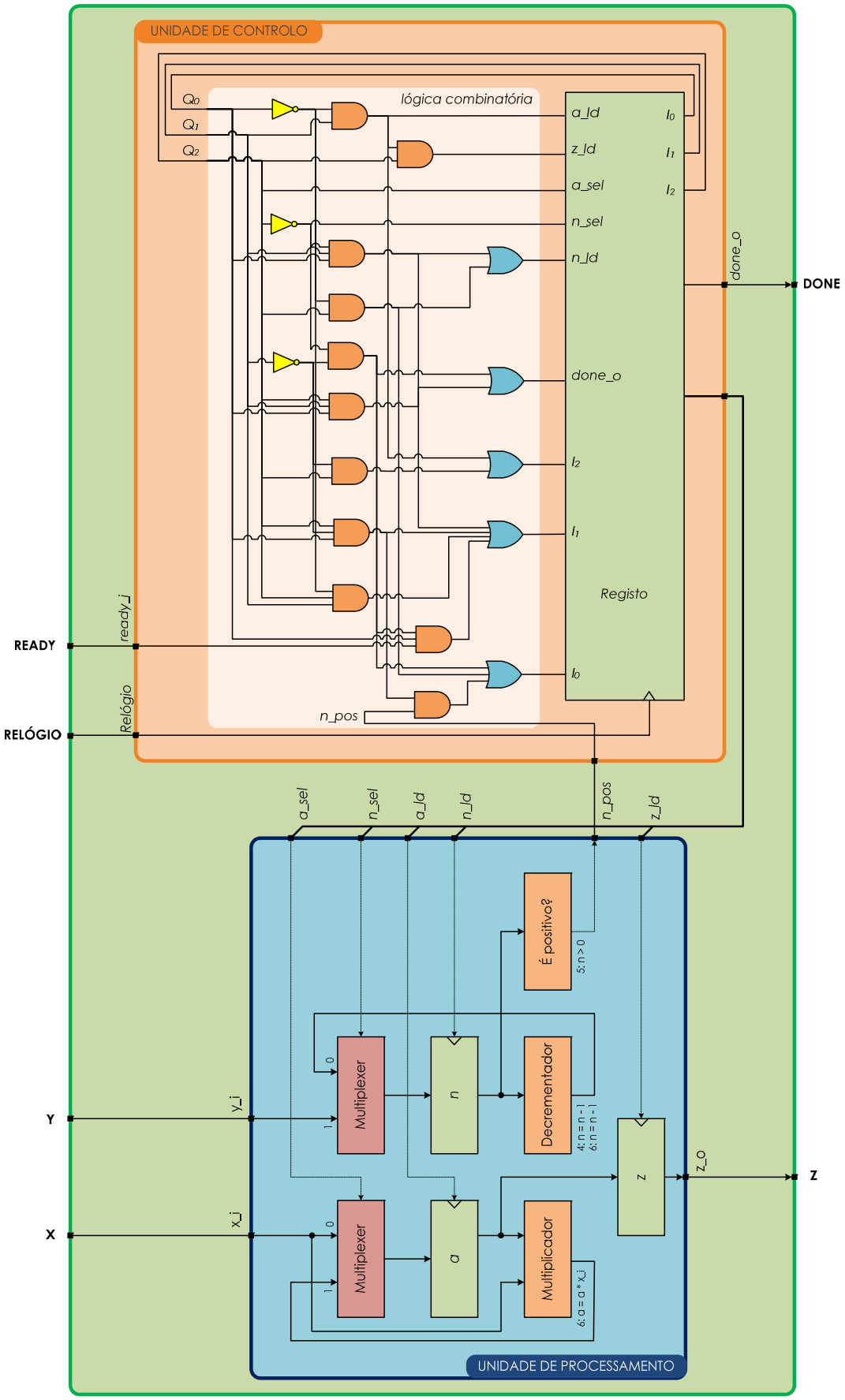


Figura 4.12 – Esquema do processador dedicado para o cálculo de x^y.

4.2 Processadores de Uso Geral

Como se viu no exemplo da secção anterior onde se projectou um processador dedicado para realização do cálculo x^y, o projecto do controlador é algo complexo tornando-se mais difícil quanto maior for o número de sinais de entrada e de saída do controlador. Em termos das funções lógicas é possível derivá-las usando ferramentas automáticas que as obtém da tabela de verdade. A implementação dessas funções usando portas lógicas e, em última análise, transístores também pode ser feita com o auxílio de um computador e o software apropriado.

Uma alternativa à implementação da lógica combinatória do controlador usando portas lógicas consiste em usar uma memória não volátil de leitura (ROM). Nessa solução os sinais de entrada, incluindo os sinais indicativos do estado da máquina de estados finita, são usados como endereço para leitura de uma palavra da memória ROM. O conteúdo dessa memória consiste nos valores que os diferentes sinais de saída devem ter para cada combinação dos sinais de entrada. No fundo é armazenado na ROM a parte da tabela de verdade do controlador que diz respeito aos sinais de saída. Dispensa-se assim a derivação das funções lógicas e a sua implementação usando portas lógicas fundamentais o que pode ser uma grande vantagem quando se pretende construir um processador usando componentes discretos.

Uma evolução desta ideia consiste em usar a memória não só para guardar a operação do controlador mas o próprio algoritmo que se pretende implementar. Usando uma memória de leitura e escrita trás ainda a vantagem acrescida de se poder facilmente alterar a função executada pelo processador. É esta a ideia por detrás de um processador de uso geral.

Obviamente que dar a liberdade ao utilizador de se poder escolher o algoritmo que o processador executa sem ter que se alterar o hardware implica que esse hardware seja capaz de realizar várias operações matemáticas que podem ou não ser utilizadas por uma determinada aplicação carregada na memória. A desvantagem que isto trás é um maior área ocupada pelo circuito electrónico do processador bem como o maior consumo energético devido à existência de mais circuitos electrónicos em funcionamento.

As operações matemáticas que se podem encontrar num processador de uso geral são as mais genéricas e abrangentes possíveis como a soma, a subtracção, a multiplicação e a divisão, várias funções lógicas (e, ou, negação, etc.) e vários tipos de deslocamento. A escolha das próprias operações matemáticas implementadas bem como o tipo de dados que são possíveis de usar (números inteiros/em vírgula flutuante ou escalares/vectores) tendo em conta o tipo de aplicações a que se destina o processador de uso geral dá origem ao aparecimento de processadores específicos (ASIP – *Application Specific Instruction-set Processor*). Exemplos desses processadores são os microcontroladores, usados em aplicações de controlo em que é importante, por exemplo, a existência de temporizadores, e processadores de sinal digitais (DSP – Digital Signal Processor) usados em aplicações que fazem uso de operações matemáticas sobre vectores de números representados em vírgula flutuante.

4.2.1 Arquitectura

Um processador de uso geral contém os mesmo dois blocos que um processador dedicado, nomeadamente a unidade de processamento (*datapath*) e a unidade de controlo. As diferenças dão-se ao nível de como são implementadas.

A unidade de processamento é constituída por uma unidade aritmética e lógica (ALU) que é capaz de executar várias operações matemáticas genéricas e por um conjunto de registos que podem ser usados para armazenar diferentes variáveis de acordo com a aplicação a ser executada (Figura 5.2).

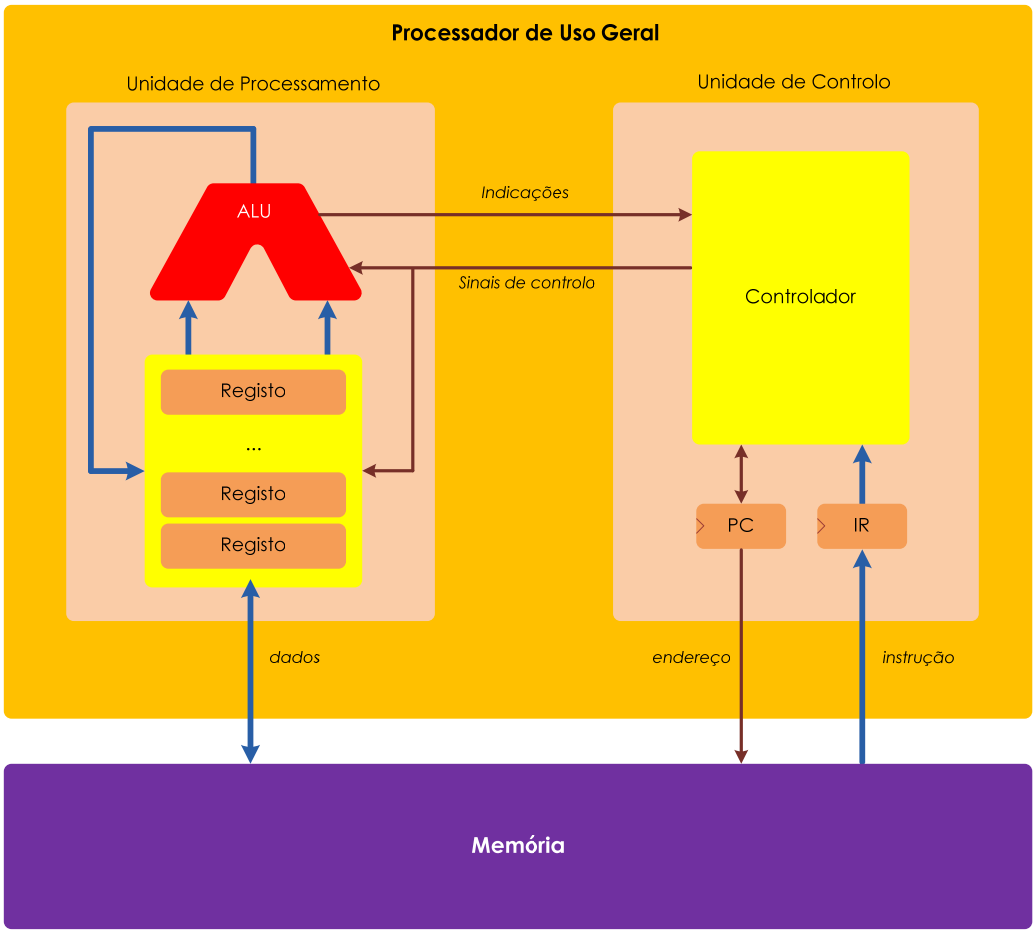


Figura 4.13 – Constituição da Unidade de Processamento e da Unidade de Controlo num processador de uso geral.

A unidade de controlo, por seu lado, possui também um controlador que no entanto não armazena o algoritmo a ser executado. Armazena sim os procedimentos necessários para buscar o algoritmo (as instruções) à memória externa, descodificar essas instruções e executá-las através do carregamento de dados da memória nos registos da unidade de processamento, controlo da ALU e armazenamento dos resultados de novo na memória. Existem assim dois registos específicos que armazenam o endereço de memória onde se encontra a instrução a ser executada (PC – *Program Counter*) e a instrução a ser executada (IR – *Instruction Register*). O PC serve para ler da memória a instrução que é então transferida para o IR. O número de bits do PC determina o espaço de memória que pode ser

ocupado por uma aplicação. Um PC com 32 bits, por exemplo, permite ter uma aplicação com 4 GB de tamanho.

Relembre-se que o computador pode ter uma arquitectura de von Neuman ou de Harvard consoante os dados e as instruções sejam guardados na mesma memória ou em memórias separadas. Um exemplo do primeiro caso (von Neuman) são computadores de secretária que usam processadores AMD ou Intel, por exemplo, e do segundo caso (Harvard) são computadores embebido (numa televisão, por exemplo) que usam microcontroladores (da Atmel ou da Microchip, por exemplo).

O número de bits que se diz ter um processador diz respeito ao comprimento das palavras processadas pela ALU, armazenadas nos registos e transferidas, através dos barramentos, para a memória. Em computadores embebidos é comum encontrar processadores com 8, 16 ou 32 bits enquanto em computadores portáteis ou de secretária é comum encontrar processadores com 32 ou 64 bits.

4.2.2 Funcionamento da Unidade de Processamento

A unidade de processamento de um processador de uso geral é semelhante à de um processador dedicado. A diferença consiste no número de circuitos usados para a realização de operações matemáticas contidos na ALU e no número de registos e sua ligação ao exterior e à ALU.

A ALU de um processador de uso geral é capaz de realizar um vasto leque de operações matemáticas. As mais comuns são as enumeradas na Tabela 4.2.

Tabela 4.2 – Lista de operações matemáticas normalmente encontradas num processador de uso geral.

Tipo	Operação
Aritmética	Soma
	Subtracção
	Multiplicação
	Divisão
	Incrementar
	Decrementar
	Comparar
	Complemento para 2
Lógica	Conjunção (AND)
	Disjunção (OR)
	Disjunção Exclusiva (XOR)
	Negação (NOT)
Deslocamento	Deslocamento Lógico para a Direita
	Deslocamento Lógico para a Esquerda
	Deslocamento Aritmético para a Direita
	Deslocamento Aritmético para a Esquerda
	Rotação para a Direita
	Rotação para a Esquerda
	Rotação para a Direita com Transporte
	Rotação para a Esquerda com Transporte

Os registos presentes na unidade de processamento são em geral em grande número (entre 30 e 100) e têm a capacidade de serem todos eles conectados às diversas entradas da ALU. O resultado de saída da ALU também é passível de ser armazenado em qualquer dos registos existentes.

Note-se que normalmente há registos que têm funções específicas e que não devem ser alterados pela aplicação. Existe normalmente um registo que a constante 0 (R0), outro registo para armazenar o endereço da pilha usada aquando de chamadas a funções para guardar os argumentos das funções, etc. Os registos específicos que contém o endereço da instrução a executar (PC) e a própria instrução a executar (IR) considera-se que fazem parte de unidade de controlo.

Naturalmente os registos têm a possibilidade de serem carregados com valores provenientes da memória e de verem os seus valores guardados em memória. O carregamento dos registos é normalmente feito no flanco ascendente do sinal que controla a gravação.

A Figura 4.14 mostra o diagrama de blocos da unidade de processamento de um processador de uso geral. O multiplexer A permite qual dos registos presente na saída do ficheiro de registos (A ou B) é colocado numa das entradas d ALU ou enviado para a memória. O multiplexer B, por sua vez, permite enviar para a ALU um dos registos do ficheiro de registos (o que estiver presente na saída B deste) ou o conteúdo do registo que guarda a instrução. Isto é usado em instruções que contêm elas próprias um dos operandos.

O multiplexer E, por sua vez, permite seleccionar o que é escrito num dos registos do ficheiro de registos: os resultados do cálculo efectuada pela ALU; dados provenientes da memória ou o conteúdo do registo de alertas da ALU. Este registo contem informação sobre a operação da ALU como, por exemplo, se o resultado é 0 ou se ocorreu um *overflow* na operação matemática.

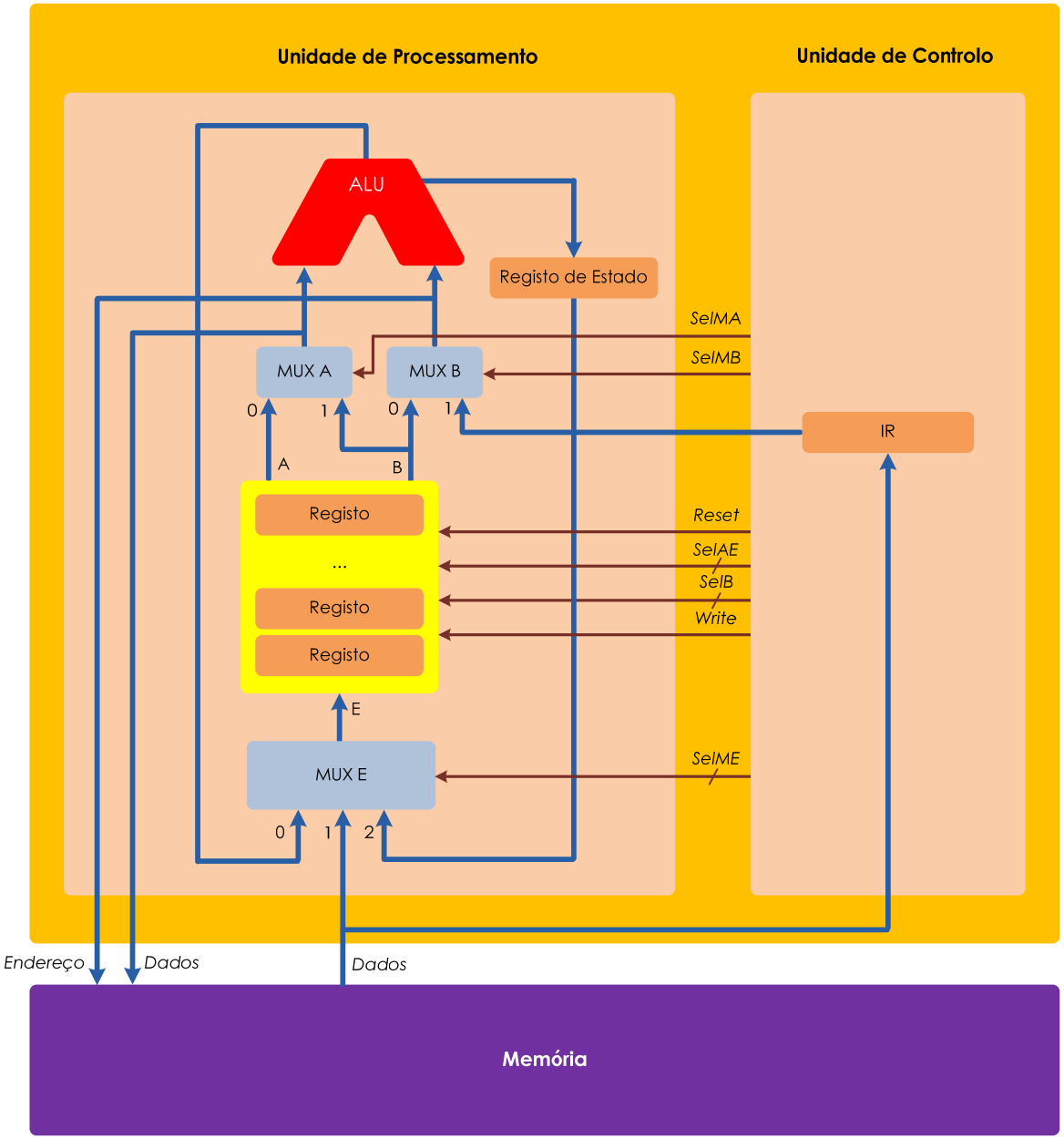


Figura 4.14 – Diagrama de blocos da unidade de processamento de um processador de uso geral.

4.2.3 Funcionamento da Unidade de Controlo

A unidade de controlo num processador de uso geral, tal como nos processadores dedicados, é responsável por configurar as operações da unidade de processamento, isto é, as operações matemáticas executadas pela ALU e a leitura/escrita nos registos.

Num processador dedicado, no entanto, essa configuração consistia no programa a executar – diferentes programas implicam diferentes unidades de controlo. Por seu lado, num processador de uso geral, o programa está armazenado na memória, onde pode ser facilmente alterado e substituído por outro. O programa já não é constituído por uma máquina de estados finita e pelas operações executadas em cada estado através da realização de funções lógicas que determinam os sinais de

controlo enviados para a unidade de processamento, mas sim por instruções genéricas a serem executadas pelo processador. O conjunto dessas instruções é predefinido e constitui o *instruction set* do processador.

A função da unidade de controlo, neste caso, é a de executar as operações determinadas pelas instruções por que é constituído o programa e que se encontram na memória do computador. Para o fazer a unidade de controlo tem que executar os seguintes 5 passos:

1) Transferir a instrução da memória do computador para um registo interno (IR – *instruction register*). O endereço de memória onde essa instrução se encontra é armazenado no seu registo interno PC (*program counter*). O valor desse registo é depois automaticamente incrementado[11] ou alterado para um valor diferente se for caso disso (instruções de salto).
2) Descodificar a instrução, isto é, determinar a partir dos 0s e 1s de que é constituída a instrução, qual a operação a executar.
3) Transferir, se necessário, os operandos das operações matemáticas a executar da memória do computador para os registos existentes na unidade de processamento.
4) Executar a operação matemática colocando os operandos nas entradas da ALU e armazenando o valor presente na saída da ALU num dos registos da unidade de processamento (passado um intervalo de tempo adequado). Algumas instruções consistem numa simples transferência de dados de uns registos para ou outros e portanto a ALU não chega a ser utilizada nesses casos.
5) Transferir o resultado da operação para a memória.

Se admitirmos que cada passo é executado num ciclo de relógio, a execução de uma instrução demora 5 ciclos de relógio. É no entanto possível, em alguns casos, executar uma instrução por ciclo de relógio se forem executados em paralelo diferentes etapas de diferentes instruções. Por exemplo, quando o processador está a executar o 2º passo de uma instrução (descodificação da instrução), é possível, ao mesmo tempo, executar o 1º passo da instrução seguinte (transferir a instrução da memória para o processador). Este tipo de funcionamento é designado por funcionamento em *pipeline*.

A execução dos 5 passos referidos pode ser encarada como a execução de um programa. Não o programa principal que o computador está a executar mas um programa simples (normalmente designado por micro-programa) cuja função é executar o programa principal armazenado na memória do computador. Assim sendo, a implementação do micro-programa pode ser feita como nos processadores dedicados, isto é, usando registos e lógica combinatória para implementar uma

[11] A memória é em geral endereçada ao byte enquanto as instruções podem ter desde 1 a 8 bytes cada. O incrementar o apontadador contido no registo PC pode significar somar, por exemplo, 4 bytes de cada vez (para instruções com 4 bytes de comprimento.

máquina de estados finita. De certo ponto de vista um processador de uso geral é um processador *dedicado* a executar um programa genérico.

Tal como no caso dos processadores dedicados ganha-se em versatilidade se em vez de se implementar a máquina de estados usando lógica combinatória se usar uma memória ROM. É assim possível, por parte do projectista, alterar facilmente o micro-programa armazenado na ROM de modo a tornar o processador mais eficiente. Uma dessas alterações, como se referiu, consiste no uso de *pipelining*. Existem no entanto muitas outras evoluções ao micro-programa básico dos 5 passos como a divisão das instruções em micro-instruções, a execução das instruções fora de ordem ou a previsão da execução futura de um programa.

5. MEMÓRIA

A memória é um componente indispensável de um computador, tanto para armazenar os dados e resultados dos cálculos efectuados como para armazenar o próprio programa. A par com o desenvolvimento do computador também os dispositivos de memória sofreram significativos avanços não só em termos de capacidade de armazenamento como em rapidez de acesso (leitura e escrita) e fiabilidade.

As tecnologias empregues para a construção de memórias são muito diversas e utilizam fenómenos físicos de praticamente todos os domínios. Os primeiros tipos de memória utilizavam fenómenos mecânicos como no caso dos cartões e fitas perfuradas ou nas memórias utilizando linhas de atraso acústicas, nomeadamente através do uso de materiais piezoeléctricos e ondas acústicas. A partir daí foram usados o campo eléctrico, no caso do tubo de Williams-Kilburn e das memórias SRAM e DRAM (Figura 5.1), o campo magnético, no caso das memórias de tambor, de núcleo magnético e nos modernos discos rígidos, e mesmo a radiação electromagnética, nomeadamente a luz visível, usada nos CDROMs, DVDs e discos BlueRay, por exemplo. Actualmente encontram-se em desenvolvimento dispositivos de memória, como as PRAM (*Phase-change Memory*) que usam efeitos térmicos para alterar o estado de um material de cristalino para amorfo e assim armazenar os bits através do estado em que se encontra o material numa certa zona.

Figura 5.1 – Fotografia de módulos de memória DRAM.

Neste capítulo começam-se por apresentar diferentes tipos de memória que foram sendo criados ao longo dos anos. O primeiro objectivo é o de utilizar o desenvolvimento de dispositivos de memória como exemplo da forma como surgem as ideias que estão por detrás das invenções que todos os dias são feitas um pouco por todo o mundo e que tem permitido o desenvolvimento tecnológico sem precedentes que se tem verificado nas últimas décadas. O segundo objectivo é o de colocar em perspectiva as características dos dispositivos de memória usados hoje em dia e como o compromisso

entre diferentes características como rapidez de acesso e capacidade de armazenamento têm impulsionado a procura de soluções cada vez mais inovadoras.

De seguida estuda-se em mais detalhe os dispositivos de memória mais utilizados hoje em dia como as SRAM e DRAM. Apresentam-se os sinais utilizados para ler e escrever a informação bem como a forma como esses dispositivos de memória são utilizados em aplicações reais.

Por fim é dedicado algum tempo à organização da memória num computador, em particular a sua hierarquia em diferentes níveis, sendo dada especial importância à estrutura e funcionamento das memórias tampão (caches).

5.1 Tipos de Memórias

Ainda mesmo antes de existirem computadores já existia a ideia de registar informação num suporte para utilização por uma máquina. Essa máquina era um tear mecânico criado por Joseph Marie Jacquard em 1801 que usava cartões perfurados[12] (Figura 5.2) que continham diferentes padrões usados para criar desenhos nos tecidos fabricados. A presença ou ausência de um furo no cartão determinava a progressão dos fios coloridos usados para criar o tecido.

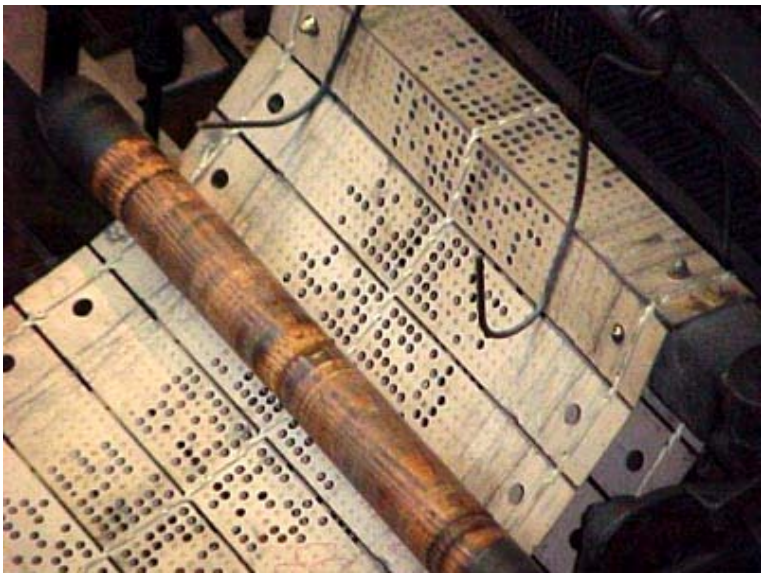


Figura 5.2 – Fotografia dos cartões perfurados usados no tear mecânico de Jacquard. Copyright by NicholasGessler 2002.

[12] Na verdade esse cartão era um rectângulo de madeira rígido.

Cada cartão continha um padrão fixo que era lido automaticamente pelo tear o que permitia a fácil criação de desenhos complicados e a diminuição do número de trabalhadores necessários para operar o tear[13]. A escrita nos cartões era feita de forma manual e permanente.

Estes cartões foram os precursores dos cartões de memória usados nos primeiros computadores como é o caso do computador mecânico projectado por Charles Babbage em 1837 e designado por *Analytical Engine*. Babbage foi o primeiro a considerar que os cartões perfurados podiam ser usados para representar uma ideia abstracta como o algoritmo ou os dados de um programa. Para além disso Babbage também teve a ideia de usar esse meio de guardar informação para armazenar o resultado intermédio de computações o que tornou possível a realização de algoritmos complexos que necessitassem de mais do que uma passagem pelos cartões perfurados (Figura 5.3).

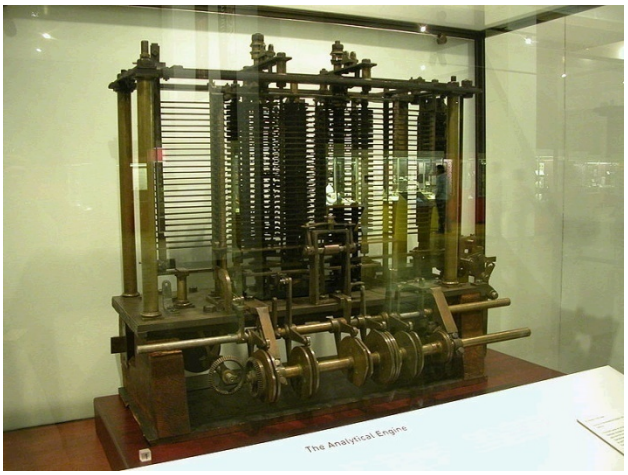


Figura 5.3 – Fotografia do Anaytical Engine projectado por Charles Babbage em 1837 e construído para o Museu da Ciência de Londres em 1992. Tirada por Bruno Barral (licença CC-BY-SA-2.5).

A mesma ideia foi desenvolvida de forma independente por Herman Hollerith em 1890 para resolver um problema muito concreto e relevante para a altura. Esse problema consistia em reduzir o tempo necessário para realizar um censo à população dos Estados Unidos da América (EUA). A constituição dos EUA obriga a que a população do país seja contada a cada 10 anos de forma a determinar a representatividade de cada Estado no Congresso[14]. Em 1880 esse censo demorou sete anos e meio. Tendo em conta o aumento da população era de prever que uma década mais tarde, em 1890

[13] Este e um dos primeiros exemplos dos efeitos sociais da tecnologia. Os trabalhadores da indústria têxtil na altura revoltaram-se pelo uso destes teares mecânicos que lhes retiravam o emprego tendo mesmo destruído alguns teares e agredido o próprio Joseph Marie Jacquard. Muitos exemplos existem onde o avanço da tecnologia levou à perda de postos de trabalho embora estudos mostrem que no geral a tecnologia tem aumentado o número de empregos.

[14] O Congresso dos EUA e o órgão legislativo do governo federal dos EUA e e constituído por duas câmaras: a Câmara dos Deputados e o Senado. Cada Estado tem direito a um certo número de deputados consoante a sua população. O Senado é constituído por 2 senadores de cada Estado, independentemente da sua população.

demorasse ainda mais tempo. Hollerith desenvolveu então uma máquina contendo um leitor de cartões perfurados, um dispositivo mecânico baseado em engrenagens para realizar a contagem e um conjunto de indicadores analógicos para indicar o resultado da soma (Figura 5.4).

Figura 5.4 – Fotografia do computador mecânico criado por Herman Hollerith para realização do censo da população nos Estados Unidos da América em 1890. Retirada de http://www.computersciencelab.com/ComputerHistory/HistoryPt2.htm.

Com base nos cartões utilizados no tear de Jacquard, que eram só de leitura, Hollerith desenvolveu cartões perfurados de leitura e escrita[15] (Figura 5.5).

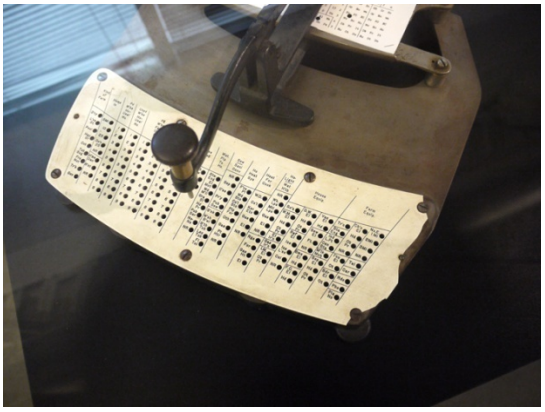


Figura 5.5 – Fotografia da máquina usada para perfurar os cartões usados para realizar o censo de 1890 nos Estados Unidos da América. Era registado o tipo de máquinas aurículas presentes, o tipo de iluminação usado em casa e o número de pessoas a viverem em determinada casa, entre outras coisas. © Australian PC Authority, 2009.

[15] Hollerith teve essa ideia ao observar que os condutores dos comboios não se limitavam a furar os bilhetes dos passageiros mas faziam-no segundo um determinado padrão que dependia das características físicas do passageiro, como a altura, peso ou cor dos olhos, por forma a evitar que esses bilhetes fossem reutilizados indevidamente por outro passageiro.

Hollerith foi bem sucedido tendo sido possível realizar o censo de 1890 em 3 anos tendo-se poupado milhões de dólares. Depois disso ele criou uma empresa, chamada de Tabulating Machine Company, que eventualmente deu origem à IBM. O principal negócio da IBM, nessa altura, era os cartões perfurados, que eram usados em inúmeras aplicações como por exemplo na facturação do gás consumido ou na determinação do valor da portagem numa estrada.

Um dos maiores problemas com o uso de cartões perfurados na altura foi o facto de serem cada vez precisos mais cartões o que levava a que facilmente saíssem da ordem. Isso foi resolvido pela utilização de uma fita de papel em vês de um conjunto de cartões.

Grace Hopper, programadora de computadores, guardou no seu livro de registos um insecto que foi encontrado, por volta de 1950, numa fita perfurada e cujas asas impediam a correcta leitura da fita pelo computador (Figura 5.6). O termo "bug" era usado, pelo menos desde 1889, para designar uma falha, no entanto foi GraceHopper que introduziu o uso do termo "debugging" para designar o processo de eliminação das falhas nos programas de computadores.

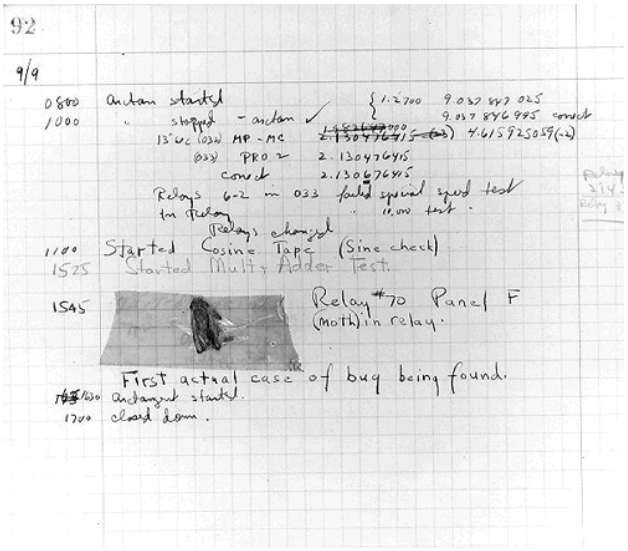


Figura 5.6 – Fotografia de uma página do diário de Grace Hopper, programadora de computadores, onde se observa um insecto que foi encontrado, por volta de 1950, numa fita perfurada e cujas asas impediam a correcta leitura da fita pelo computador. Copyright IEEE, 2002.

5.1.1 Usando Som para Guardar Informação – Linha de Atraso Acústica

Há medida que os computadores deixaram de ser mecânicos para passarem a ser electrónicos, em meados do século XX, também o tipo de memória utilizado evolui dos cartões e fitas perfuradas para dispositivos electrónicos. Uma das primeiras memórias electrónicas foi a linha de atraso acústica criada por J. Presper Eckert por volta de 1943 aquando do desenvolvimento do computador electrónico EDVAC.

A evolução do tipo de memorias electrónicas desenvolvidas e determinada, tal como acontece em outras áreas, por dois factores: as ideias inovadoras que os engenheiros têm e as capacidades de fabricação existentes. No caso da memória utilizando linhas de atraso acústicas, Eckert teve a ideia

para elas a partir do trabalho que tinha desenvolvido no passado em radares. Ai ele teve a necessidade de eliminar os ecos provocados por objectos fixos e que dificultavam a detecção dos objectos móveis (aviões). Era então necessário eliminar os sinais recebidos pela antena que demorassem sempre o mesmo tempo a ir e vir. A ideia que surgiu na altura foi a de subtrair dos sinais recebidos pela antena uma copia do sinal recebido no impulso anterior. Punha-se então o problema de como criar essa copia. Tal foi resolvido com uma linha de atraso. O sinal recebido pela antena era atrasado de um intervalo de tempo igual ao espaçamento entre os impulsos enviados. Assim, os sinais saiam da linha de atraso no exacto instante em que o eco correspondente ao impulso seguinte estava a chegar a antena. Subtraindo os dois sinais era possível eliminar os ecos correspondentes a alvos fixos e assim apresentar no ecrã unicamente a informação correspondente aos alvos moveis. A linha de atraso usava ondas acústicas por propagarem-se muito mais devagar do que as ondas electromagnéticas e assim ser possível ter linhas de atraso mais curtas ($\Delta t = \Delta x/v$).

A mesma ideia foi portanto usada por Eckert para criar memórias electrónicas. A informação era representada de forma binária utilizando impulsos eléctricos. A presença de um impulso, correspondente a um "1", levava a criação de uma onda acústica que era transmitida de um dos lados da linha de atraso por um transdutor piezoeléctrico e recebida no outro extremo, também por um transdutor piezoeléctrico que convertia a onda acústica de novo num sinal eléctrico. Esse sinal eléctrico era então amplificado e enviado para o outro extremo da linha para ser de novo introduzido nesta (Figura 5.7). Para manter a informação armazenada era preciso continuamente criar as ondas acústicas, isto e, a informação tinha que ser regenerada. Obviamente que neste caso a informação perdia-se caso a alimentação do circuito electrónico de regeneração fosse cortada. A memória baseada na linha de atraso acústica é portanto volátil.

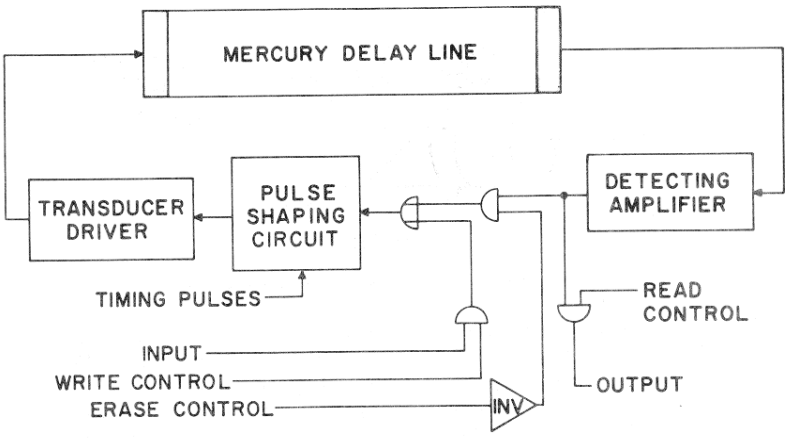


Figura 5.7 – Diagrama de blocos de uma memoria do tipo Linha de Atraso Acústica.

Este tipo de memoria é capaz de armazenar vários bits através de um padrão de "0" e "1" que flui constantemente na linha de atraso. A divisão do tempo de propagação pelo número de bits determina quanto tem que ser o intervalo entre os vários bits.

As primeiras memórias deste tipo usava como linha de atraso um tubo contendo mercúrio. Este material foi escolhido por ter uma impedância acústica semelhante à dos próprios transdutores

piezoeléctricos o que diminuía substancialmente as perdas por reflexão devido a desadaptação de impedância acústica. O mercúrio era mantido a uma temperatura de 40º C para melhorar a adaptação de impedâncias.

Além de ser volátil, este tipo de memória é de acesso serie ao contrário das memórias actuais de acesso aleatório. O tempo de acesso máximo (pior caso) é determinado pelo tempo de propagação das ondas acústicas na linha de atraso e que é da ordem do milissegundo.

5.1.2 Memória no Ecrã de um Televisor – Tubo de Williams

Outro exemplo de uma tecnologia existente ser usada para outro fim é o caso do receptor de televisão e como o seu dispositivo de base, o tubo de raios catódicos, foi usado para criar uma memoria para computador. Isso aconteceu em 1946 quando Freddie Williams começou a desenvolver o que mais tarde viria a ser chamado de Tubo de Williams. Na verdade este tipo de memória é também chamado de Tubo de Williams-Kilburn pois Williams só conseguiu armazenar 1 bit antes de ter mudado de emprego do Telecommunications Research Establishment (TRE) para a Universidade de Manchester. Foi Tom Kilburn que continuou o desenvolvimento da memória baseada no tubo de raios catódicos tendo, em 1947, conseguido armazenar 2048 bits durante algumas horas.

O princípio de funcionamento é bastante simples. Um tubo de raios catódicos contém uma fonte de electrões que produz electrões que são depois acelerados em direcção a um dos extremos do tubo (ecrã) que está coberto de fósforo de modo que quando é atingido por electrões liberta luz visível. Fazendo varrer o feixe de electrões ao longo do ecrã e variando a intensidade desse feixe é possível criar uma imagem visível (a preto e branco no caso mais simples).

Os electrões que embatem no ecrã provocam a libertação de electrões por um processo chamado de *emissão secundária*. Como resultado é possível criar zonas de carga positiva no ecrã. Dividindo este em diferentes áreas separadas é possível criar uma matriz de bits que são representados pela quantidade de carga com que fica o ecrã. Uma placa metálica do lado de fora do ecrã é usada para ler os valores dos bits. A Figura 5.8 apresenta uma fotografia de uma destas memórias.

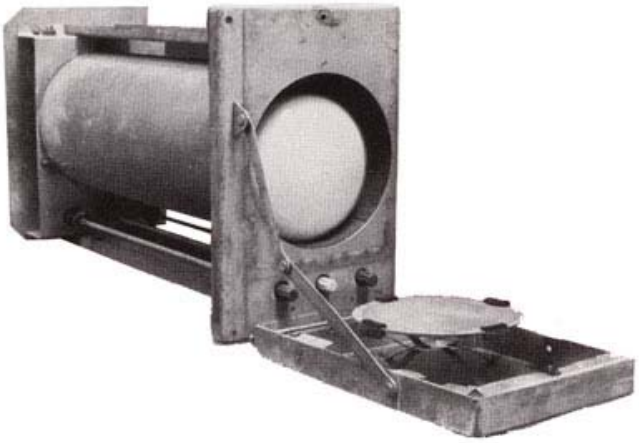


Figura 5.8 – Fotografia de uma memória do tipo Tubo de Williams. Licença Creative Commons Attribution ShareAlike 3.0.

Como a carga acumulada no ecrã dispersa-se rapidamente é necessário estar constantemente a refrescar a carga armazenada em cada bit. Este tipo de dispositivo de memória, tal como a linha de atraso acústica, é portanto volátil. O Tubo de Williams-Kilburn tem no entanto a vantagem de ser um ter um acesso aleatório e não sequencial. É também mais rápido, feito com componentes comuns e não necessita de um controlo apartado de temperatura ou da tensão de alimentação.

A ideia de usar como suporte de informação algo que só consegue reter a informação por breves milissegundos e a consequente necessidade de regeneração periódica dessa informação é usada hoje em dia em memórias do tipo DRAM e que se encontram em praticamente todos os computadores pessoais.

5.1.3 Memória de Tambor

Com o advento dos computadores utilizando a arquitectura de von Neumann, em que tanto o programa como os dados são armazenados no mesmo dispositivo de memória, surgiu a necessidade de memórias com maior capacidade de armazenamento. A memória de tambor (*drum memory*) foi uma das soluções encontradas (Figura 5.9). Inventada em 1932 por Gustav Tasuschek, foi muito utilizada nas décadas de 50 e 60. Este tipo de dispositivo consiste num tambor rotativo coberto de um material ferromagnético. À volta do tambor eram dispostas as cabeças de leitura e escrita. Cada cabeça correspondia a uma pista diferente no tambor ao contrário do que acontece hoje em dia nos discos rígidos utilizados na maioria dos computadores de secretaria e portáteis onde as cabeças são posicionadas na pista desejada por acção de um motor passo-a-passo. Outra diferença entre a memória de tambor e os discos rígidos modernos é que nestes últimos o material ferromagnético é disposto em discos planos e não à volta de um tambor.

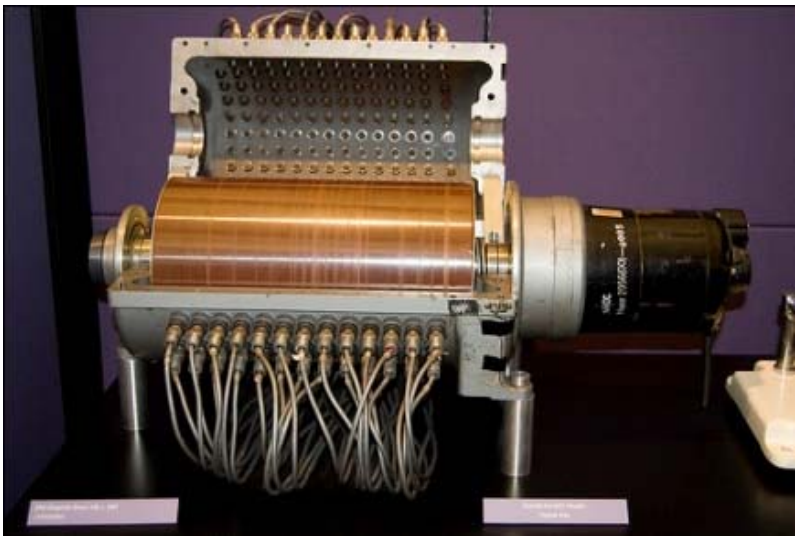


Figura 5.9 – Fotografia de uma memória de tambor onde se observa o tambor rotativo coberto por um material ferromagnético e as cabeças de leitura e escrita. © Flickr.

O tempo de acesso à informação na memória de tambor depende unicamente da sua velocidade de rotação ao contrário dos discos rígidos modernos em que depende também do tempo necessário para posicionamento da cabeça de leitura/escrita.

Um dos primeiros computadores onde este tipo de memória foi utilizado foi o Manchester Mark I em 1949. Este computador, baseado na arquitectura de von Neumann, utilizava não só memória de tambor mas também tubos de Williams-Kilburn cujo tempo de acesso era menor.

5.1.4 Cada Bit num Anel Magnético – Memória de Núcleo de Ferrite

As memórias de núcleo de ferrite, desenvolvidas no início da década de 50, também usam o campo magnético para armazenar informação. Neste caso são usados pequenos anéis de ferrite que são magnetizados segundo duas direcções opostos para assim armazenarem um "0" ou um "1".

Os anéis de ferrite são dispostos segundo uma grelha feita de fios condutores. Cada anel é colocado na intersecção desses fios (Figura 5.10). Estas memórias em geral utilizavam várias grelhas para armazenar os diferentes bits de uma mesma palavra. Cada palavra podia assim ser lida ou escrita em paralelo actuando-se em cada grelha ao mesmo tempo.

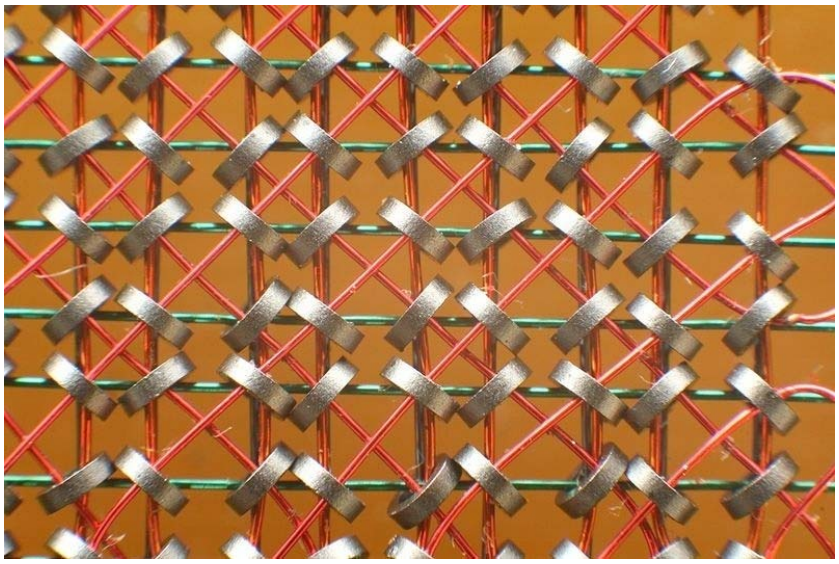


Figura 5.10 – Fotografia dos anéis de ferrite e das linhas de leitura e escritas de uma memória de núcleo de ferrite. Cada anel tem 1 mm de diâmetro.

Para mudar a polaridade do campo magnético de um dado anel é necessário aplicar um campo magnético com um valor superior a um certo limite. Isso é feito colocando no condutor horizontal e no condutor vertical uma corrente com metade do valor necessário para se alterar a magnetização do anel. Assim só o anel que se encontra na intersecção desses dois condutores terá a sua polarização alterada porque todos os outros que se encontram na mesma linha ou na mesma coluna terão só metade do campo necessário. Esta é uma forma engenhosa de se aceder aos anéis sem ter que se usar um condutor para cada um. A escolha do sentido da corrente determina qual o sentido da polarização imposta no anel.

A leitura do bit armazenado numa determinada posição é feita determinando-se o sentido do campo magnético do anel correspondente. Isso consegue-se fazer escrevendo nesse local o bit "0". Se esse anel já se encontrasse com o sentido de polarização correspondente a um "0", nada aconteceria. Se, pelo contrário, esse bit fosse "1", então o sentido do campo sofreria alteração o que era iria induzir

uma corrente num terceiro fio condutor (denominado de "Sense") que atravessa os anéis ao longo das diagonais da grelha. Portanto, a detecção de uma corrente nesse condutor significa que estava armazenado um "1" e a não detecção de corrente significa que estava armazenado um "0". Em ambos os casos, depois da leitura, o bit armazenado seria um "0" o que provoca uma destruição da informação armazenada. Para evitar isso, cada operação de leitura é seguida de uma operação de escrita que repõe o valor do bit lido caso seja "1".

As primeiras memórias deste tipo tinham um tempo de acesso de 12 μs, o que já por si era muito melhor do que as memórias existentes. Em meados da década de 70 esse valor tinha sido reduzido para 1,2 μs.

Este tipo de memória é não-volátil pois não é necessária alimentação para manter a polarização magnética dos anéis de ferrite.

5.1.5 "Perfurando" um Circuito Integrado – Memória de Máscara

Até agora, com excepção dos cartões e fitas perfuradas, os dispositivos de memória referidos possibilitam a leitura e a escrita. Estas primeiras formas de memória eram usadas para armazenamento de dados e portanto tinham mesmo de possibilitar a leitura e escrita de informação. Com o aparecimento de computadores que armazenavam tanto o programa como os dados em memória surgiu a possibilidade de se usarem memórias só de leitura para armazenar o programa. A principal vantagem dessas memórias é a sua simplicidade de construção, sendo portanto mais baratas. Têm no entanto a desvantagens de terem de ser descartadas caso se queira alterar o programa.

Os cartões e fitas perfuradas são um tipo de memória só de leitura, no entanto, a partir da década de 40 e 50, com o advento dos computadores electrónicos, esse tipo de memória, criado para trabalhar com computadores mecânicos, não era adequado. Surgiu então uma memória electrónica só de leitura inspirada, no fundo, pelos cartões perfurados. De entre todos os tipos de memória vistos até agora, esse era o único que se baseava no armazenamento da informação no próprio material de que era feito a memória (cartão, nesse caso). A falta ou não de material no cartão, isto é, a existência ou não de um furo, representava um "0" ou um "1". No caso dos restantes tipos de memória apresentados, a informação era armazenada utilizando uma *propriedade* do material. As linhas de atraso utilizavam a pressão do mercúrio dentro do tubo (onda acústica), o tubo de Williams-Kilburn usava a carga eléctrica armazenada no ecrã do tubo de raios catódicos e a memória de tambor e de núcleo de ferrite usavam o valor do campo magnético.

As primeiras memórias electrónicas de computador só de leitura voltavam a usar o mesmo princípio – o do armazenamento da informação através da presença ou ausência de material, só que desta vez aplicado a um circuito electrónico o que era obviamente apropriado para ser usado num computador electrónico. O exemplo mais antigo deste tipo de memórias, desenvolvido na década de 60, chamado de Memória de Máscara (*Mask ROM*), consistia em dois conjuntos de condutores: um para o endereço da posição de memória e outro para a saída dos dados armazenados. O símbolo "1" era então armazenado através da ligação de um díodo entre a linha de dados correspondente ao bit em causa e a linha de endereço correspondente à posição de memória a ser lida (Figura 5.11).

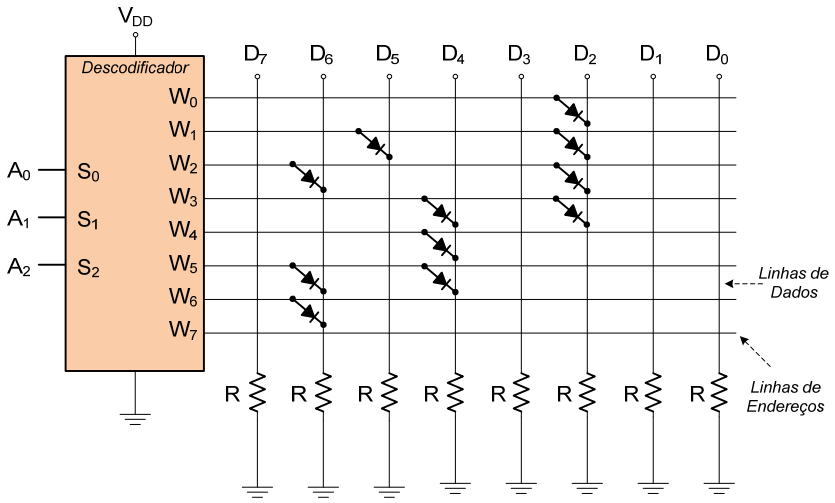


Figura 5.11 – Ilustração da constituição de uma memória de Máscara feita com componentes discretos.

Quando uma das linhas de endereços está no nível alto (W_0, por exemplo) a tensão V_{DD} é aplicada aos diodos que a ela estão ligados. Esses diodos ficam a conduzir impondo a tensão V_{DD} à linha de dados a que estão ligados (D_2, no caso do exemplo), o que corresponde a um "1". Nas linhas de dados que não têm nenhum diodo ligado, a tensão permanece zero devido à resistência de ***pull-down*** que cada uma tem.

Com o advento dos circuitos integrados este tipo de memória passou a ser realizada usando transístores em vez de díodos. Também as resistências de pull-down são substituídas por transístores de pull-up pois os transístores ocupam menos espaço no circuito integrado (Figura 5.12).

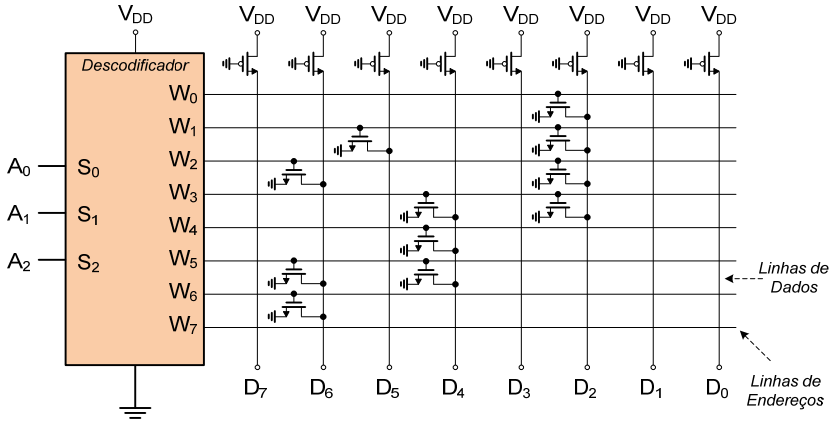


Figura 5.12 – Ilustração da constituição de uma memória de Máscara integrada.

Naturalmente que os dados armazenados neste tipo de memória são determinados na altura da fabricação, não podendo depois ser alterados. O nome deste tipo de memória advém exactamente das máscaras que são usadas para fazer o circuito integrado utilizando técnicas de litografia. Estas são fabricadas tendo em conta os transístores que se deseja e a sua localização. Pelo facto de ser caro fabricar as máscaras, o preço deste tipo de memórias só é competitivo se forem produzidas dezenas de milhar de circuitos integrados idênticos.

A Memória de Máscara é portanto um tipo de memória que não é usado durante o desenvolvimento de um novo produto. Só quando o seu desenho está concluído e se deseja então iniciar a produção em larga escala é que se passa a usar este tipo de memória que é hoje em dia principalmente encontrada em microcontroladores para armazenar o programa inicial que corre assim que este é ligado (procedimento conhecido por "booting"). Antigamente era usado também em computadores compatíveis com o IBM/PC para armazenar a BIOS (*Basic Input/Output System*) que é o programa que corre inicialmente no computador para inicializar os periféricos como a placa de vídeo e os discos rígidos, por exemplo.

Outras aplicações tradicionais deste tipo de memórias são as tabelas para cálculo numérico. No caso das funções trigonométricas, por exemplo, eram muitas vezes guardadas tabelas com alguns valores pré-calculados sendo depois feita interpolação consoante o valor desejado. Como o conteúdo dessas tabelas nunca muda, são uma boa aplicação para as memórias Mask ROM. Uma outra aplicação, já muito em desuso, era a utilização em placas gráficas para armazenar o desenho dos caracteres usados para representar texto. Naturalmente que isso significava, em geral, que o tipo de letra não podia ser alterado.

A Figura 5.13 mostra a fotografia de um peixe robótico que canta quando alguém se aproxima. Possui uma memória Mask ROM que armazena o programa para controlo dos motores que fazem a cabeça e barbatana mexer bem como 30 segundos de música.

Figura 5.13 – Fotografia de um peixe robótico que canta quando alguém se aproxima. Possui uma memória Mask ROM que armazena o programa para controlo dos motores que fazem a cabeça e barbatana mexer e também 30 segundos de música. Tirada por "RED Tuna". Licença Creative Commons Attribution (CC-BY).

5.1.6 Cada Fusível, Cada Bit – PROM

As memórias PROM (*programmable read-only memory*) utilizam a mesma ideia das memórias de Máscara mas têm a vantagem de poderem ser escritas uma vez depois de construídas. Isso é conseguido usando fusíveis em vez de transístores. A memória sai da fábrica com todos os bits a "1". Durante a escrita os zeros são armazenados fundindo determinados fusíveis através do uso de uma tensão elevada.

A Figura 5.14 mostra a fotografia de um circuito integrado fabricando em 1983 contendo uma memória PROM e usado no computador Sinclair 48k ZX Spectrum.

Figura 5.14 – Fotografia de um circuito integrado fabricando em 1983 contendo uma memória PROM e usado no computador Sinclair 48k ZX Spectrum. Tirada por Bill Bertram.Licença Creative Commons Attribution (CC-BY).

Este tipo de memória foi inventado em 1956 por Wen Tsing Chow a pedido da força aérea dos Estados Unidos da América que procurava uma forma fácil de armazenar algumas constantes no computador Atlas E/F ICBM usado em mísseis balísticos intercontinentais.

5.1.7 EPROM, EEPROM e Flash

As memórias EPROM (*erasable programmable read only memory*), EEPROM (*electrically erasable programmable read only memory*) e Flash são memórias não voláteis. Este tipo de memória é baseado em transístores de efeito de campo com porta flutuante (FGMOS).

Um transístor de efeito de campo com porta flutuante tem uma constituição semelhante a um transístor de efeito de campo. Possui, no entanto, um material condutor entre a porta e o subtrato – porta flutuante, como se observa na Figura 5.15. Não existe nenhuma ligação eléctrica com a porta flutuante, ao contrário do que acontece com a porta, o substrato, a fonte e o dreno.

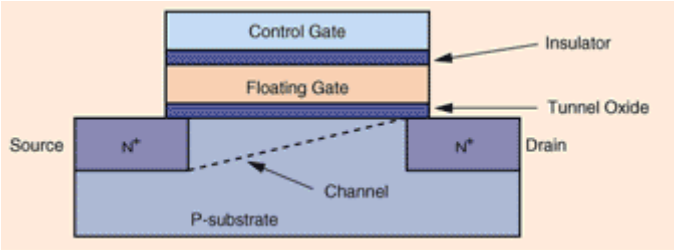


Figura 5.15 – Constituição de um transístor de efeito de campo com porta flutuante.

O armazenamento de um bit de informação é feito através de armazenamento de electrões na porta flutuante. A presença de electrões corresponde ao bit "0" e a ausência ao bit "1".

A diferença entre os três tipos de memória tem a ver com a forma como são programadas e apagadas (Tabela 5.1).

Tabela 5.1 – Métodos usados para programação e limpeza de memórias EPROM, EEPROM e Flash.

Tipo	Programação	Limpeza
EPROM	Injecção de Portadores Quentes	Luz Ultravioleta
EEPROM	Efeito de Túnel	Efeito de Túnel

Flash	Injecção de Portadores Quentes	Efeito de Túnel

A programação pela injecção de portadores quentes é feita através da aplicação de uma tensão elevada entre o dreno e a fonte (tipicamente 7 V) o que faz com que fluam electrões entre a fonte e o dreno com elevada energia cinética – portadores quentes. Ao mesmo tempo é aplicada uma tensão entre a porta e o substrato (tipicamente 12 V) de forma a direccionar esses electrões para a porta flutuante onde são injectados depois de atravessarem a camada dieléctrica que geralmente é feita de dióxido de silício (Figura 5.16). Ao atravessar o dieléctrico os electrões provocam a sua degradação o que limita o número de programações que é possível fazer.

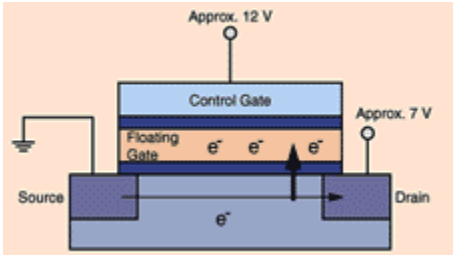


Figura 5.16 – Ilustração da programação numa memória baseada em transístores de efeito de campo com porta flutuante.

A remoção dos electrões presentes na porta flutuante (limpeza) nas memórias EPROM é feita através da incidência de luz ultravioleta a qual provoca ionização dos átomos do dieléctrico que acabam por absorver os electrões da porta flutuante. A Figura 5.17 mostra a fotografia de uma memória EPROM com a janela feita de quartzo através da qual a luz ultravioleta pode incidir no *chip*.

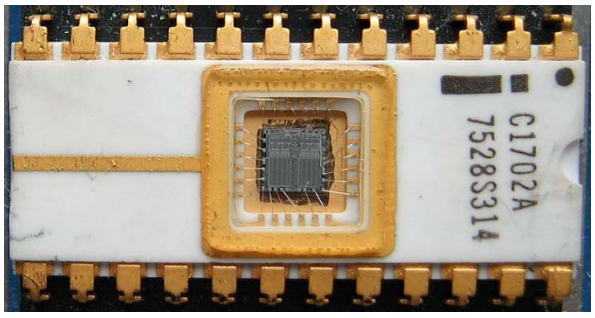


Figura 5.17 – Fotografia de uma memória EPROM. Disponibilizada na Wikipedia. Tirada por Poil. Licença Creative Commons Attribution ShareAlike 3.0.

A remoção dos electrões presentes na porta flutuante (limpeza) pode ser conseguida usando o efeito de túnel. Isso é feito através da aplicação de uma tensão de -9 V à porta e de 6 V à fonte (Figura 5.18).

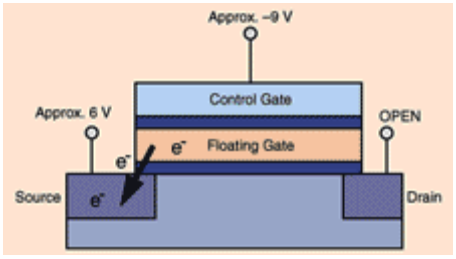


Figura 5.18 – Ilustração da escrita de um "1" numa memória baseada em transístores de efeito de campo com porta flutuante.

A leitura do valor do bit armazenado é feita através da aplicação de uma tensão de 5 V à porta e de 1 V ao dreno. Se houver corrente entre a fonte e o dreno então é porque existem electrões na porta flutuante e o bit armazenado é "0" (Figura 5.19). Caso contrário o bit armazenado é "1".

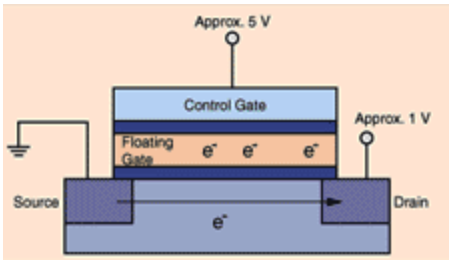


Figura 5.19 – Ilustração da escrita da leitura do valor de um bit armazenado numa memória baseada em transístores de efeito de campo com porta flutuante.

Quando nenhuma tensão está aplicada ao transístor o valor da carga na porta flutuante mantém-se constante e a informação é preservada.

As memórias EEPROM são de acesso aleatório sendo possível ler e escrever um único byte de cada vez. As memórias flash, no entanto, só permitem a escrita de um conjunto de bytes. Isso permite a eliminação de alguma da lógica de endereçamento o que aumenta a capacidade de armazenamento.

5.1.8 SRAM

Hoje em dia, como se viu, o processamento de informação levada a cabo pelos computadores é feita por circuitos electrónicos digitais baseados em transístores do tipo MOSFET. Não é de estranhar, portanto, que o mesmo seja usado para o armazenamento de informação. Isso, no entanto, só é possível em memórias voláteis, ou seja, naquelas em que a informação só é guardada enquanto o circuito electrónico estiver alimentado. De entre estas memórias voláteis destacam-se dois tipos básicos: memórias estáticas, designadas por SRAM (*static RAM*) e memórias dinâmicas, designadas por DRAM (*dynamic RAM*). No primeiro caso a informação, depois de escrita, é mantida na memória enquanto esta estiver alimentada, enquanto no segundo caso a informação armazenada tem de ser periodicamente reescrita na memória de forma a não se perder.

As memórias SRAM armazenam os bits de informação num sinal eléctrico de tensão que percorre o mesmo circuito electrónico de forma circular. Isso é feito com a ajuda de duas portas lógicas NOT

como ilustrado na Figura 5.20. A saída de cada porta está ligada à entrada da outra. Existem assim dois pontos no circuito, Q e Q´, que têm sempre níveis lógicos diferentes. Esta célula básica armazena um 1 quando a tensão no ponto Q é alta e no ponto Q´ é baixa, e armazena um 0 quando a tensão no ponto Q é baixa e no ponto Q' é alta. Este tipo de circuito é o mesmo usado em flip-flops.

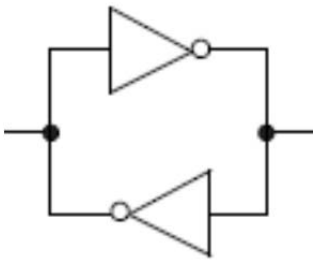


Figura 5.20 – Ilustração do circuito básico de armazenamento de um bit numa célula de memória SRAM.

Tendo em conta a função de transferência de uma porta lógica NOT apresentada na Figura 5.21, é possível determinar os pontos de funcionamento possíveis para este circuito (Figura 5.22).

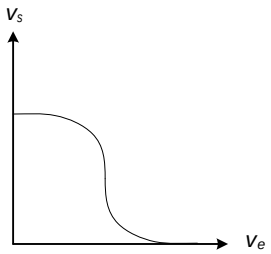


Figura 5.21 – Função de transferência típica de uma porta lógica NOT.

Destes 3 pontos de funcionamento só dois são estáveis – O A e o B sendo usados para representar o "0" e o "1".

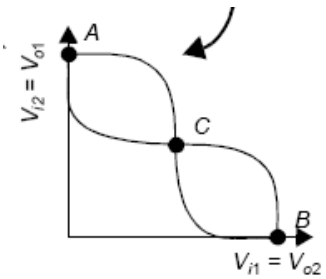


Figura 5.22 – Pontos de funcionamento da célula de memória SRAM.

As duas portas NOT são em geral implementadas usando lógica CMOS como ilustrado na Figura 5.23.

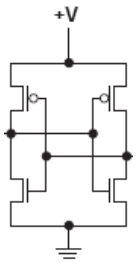


Figura 5.23 – Implementação em lógica CMOS do circuito bi-estável usado na memória SRAM.

Este tipo de memória é, hoje em dia, maioritariamente usada como memórias tampão em processadores devido à sua rapidez de acesso. É uma memória volátil facilmente construída usando as técnicas típicas de construção de circuitos integrados.

5.1.9 DRAM

A memória DRAM (*dynamic random access memory*) armazena informação como carga num condensador. O acesso a essa informação é feito através de um transístor MOSFET como ilustrado na Figura 5.24.

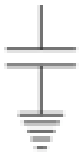


Figura 5.24 – Célula de memória DRAM.

Devido a correntes de fuga a carga armazenada no condensador depressa desaparece. Esse problema é resolvido através do constante refrescamento da informação que consiste no recarregamento periódico do condensador (no caso do armazenamento de um "1").

Devido ao menor número de componentes de que é constituída a célula de memória DRAM conseguem-se construir memórias deste tipo com maior capacidade do que as memórias SRAM sendo hoje em dia usadas como a memória principal em computadores portáteis e de secretária. As memórias DRAM têm, no entanto, a desvantagem de terem um tempo de acesso maior, devido à necessidade de efectuar a carga e descarga do condensador.

São da mesma família as memórias conhecidas como SDRAM (*synchronous* DRAM). Essas memórias têm a mesma célula de armazenamento do que as memórias DRAM (Figura 5.24) mas o seu funcionamento é síncrono com um sinal de relógio externo. Outra variante deste tipo de memória são as memórias DDR SDRAM que permitem a leitura e escrita de informação nos dois flancos de relógio o que duplica a largura de banda.

5.1.10 Comparação

A Tabela 5.2 apresenta as características mais importantes dos diferentes tipos de memória usados hoje em dia.

Tabela 5.2 – Comparação dos diferentes tipos de memória modernas [2].

Caracterísica	ROM					RAM	
	Mask ROM	PROM	EPROM	EEPROM	Flash	SRAM	DRAM
Custo	Muito Baixo	Baixo	Médio	Médio	Baixo	Alto	Baixo
Velocidade	Alta	Alta	Média	Baixa	Média	Muito Alta	Alta
Programável	Não	Uma vez	Sim	Sim	Sim	Sim	Sim
Volátil	Não	Não	Não	Não	Não	Sim	Sim
Consumo	Muito Baixo	Baixo	Baixo	Médio	Baixo	Baixo	Médio

5.2 Sistema de Memória

5.2.1 Organização das Células

As memórias modernas usadas hoje em dia em computadores, quer sejam SRAM ou DRAM, são constituídas por células idênticas de 1 bit organizadas de forma matricial. Isso permite reduzir o número e tamanho dos condutores necessários para aceder a cada célula, isto é, a cada bit.

A mesma ideia era já utilizada nas memórias de anéis de ferrite onde os bits eram armazenados através do sentido de magnetização desses anéis. A leitura e escrita da informação necessitavam da aplicação de uma corrente eléctrica suficiente grande para alteração do sentido da magnetização. De forma a reduzir o número de fios necessário esses anéis eram organizados de forma matricial, passando por cada anel dois fios perpendiculares. Para aceder a um desses anéis era colocada metade da corrente no condutor vertical e metade no condutor horizontal que por ele passavam.

Nas memórias modernas, parte dos bits de endereço são usadas para activar, com um descodificador, uma linha de células de memória. A célula desejada é então seleccionada com um multiplexer ligado às diferentes colunas da matriz e controlado pelos restantes bits de endereço, como se ilustra na Figura 5.25.

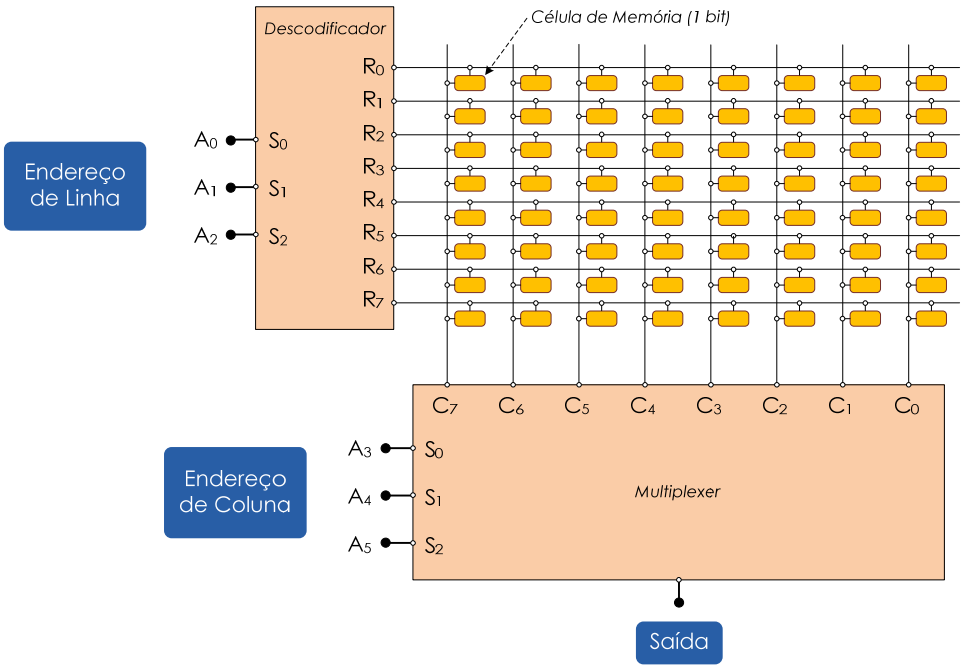


Figura 5.25 – Exemplo da organização matricial das células de memória numa memória de 64 bits.

De forma a minimizar o número de fios de ligação a matriz deve ser quadrada, isto é, ter o mesmo número de linhas e de colunas. No caso de uma memória com 1 Mbit de capacidade, por exemplo, deve-se ter 1024 linhas e 1024 colunas.

Na realidade a leitura e escrita da informação na memória não é feita por bits mas sim por palavras que geralmente têm 8, 16 ou 32 bits de largura. Essas palavras são constituídas por bits armazenados em células ligadas à mesma linha de forma tornar a sua leitura e escrita mais rápida.

Tendo em conta que a matriz deve ser aproximadamente quadrada, em cada linha existirão várias palavras armazenadas. No exemplo, anterior – o de uma memória com 1 Mbit de capacidade – e considerando palavras de 8 bits, cada uma das linhas tem 128 palavras (1024/8) perfazendo um total de 131072 palavras armazenadas na memória (128 kbytes).

A Figura 5.26 mostra o exemplo de uma memória de 64 bits com palavras de 2 bits. Existem, neste caso, dois multiplexers que selecionam as colunas de cada um dos bits da palavra de saída. Esta memória tem portanto a capacidade de 32 palavras (8 linhas e 4 palavras por linha).

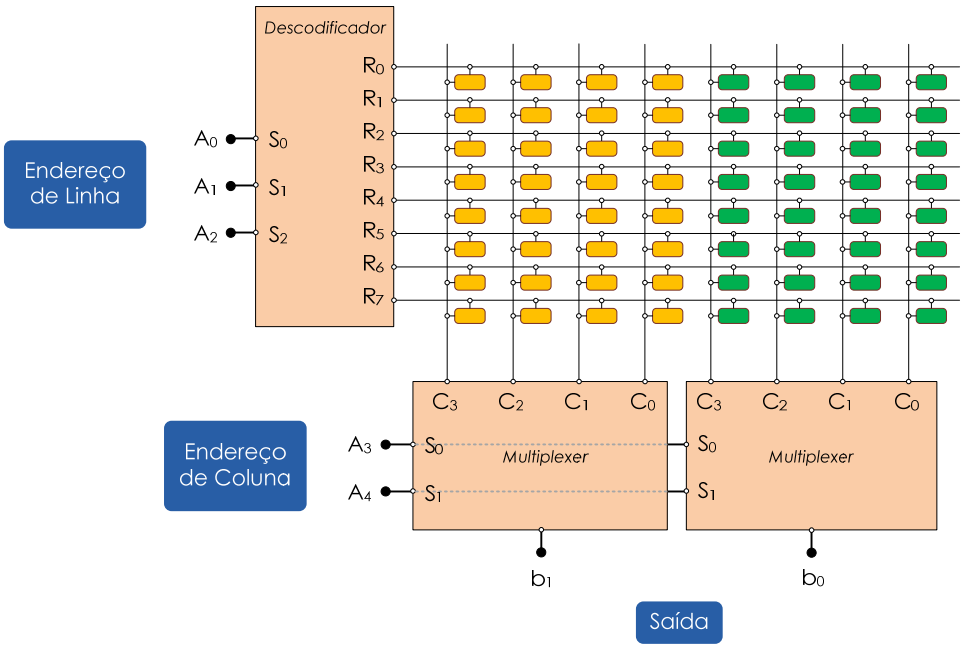


Figura 5.26 – Exemplo da organização matricial das células de memória numa memória de 64 bits com palavras de 2 bits.

5.2.2 Barramentos

Para minimizar o número de ligações ao circuito integrado que contém a memória é comum usar as mesma linhas para enviar o endereço da linha e da coluna. Sinais de controlo são usados para indicar qual dos endereços está presente no barramento de endereços e são usados dois registos para manter os endereços de linha e coluna. A Figura 5.27 ilustra o caso em que é usado um barramento de endereços com 3 linhas e um de dados com duas linhas. As linhas de controlo RAS (*row address strobe*) e CAS (*column address strobe*) determinam o armazenamento do endereço presente no barramento no registo de endereço de linha e de coluna respectivamente.

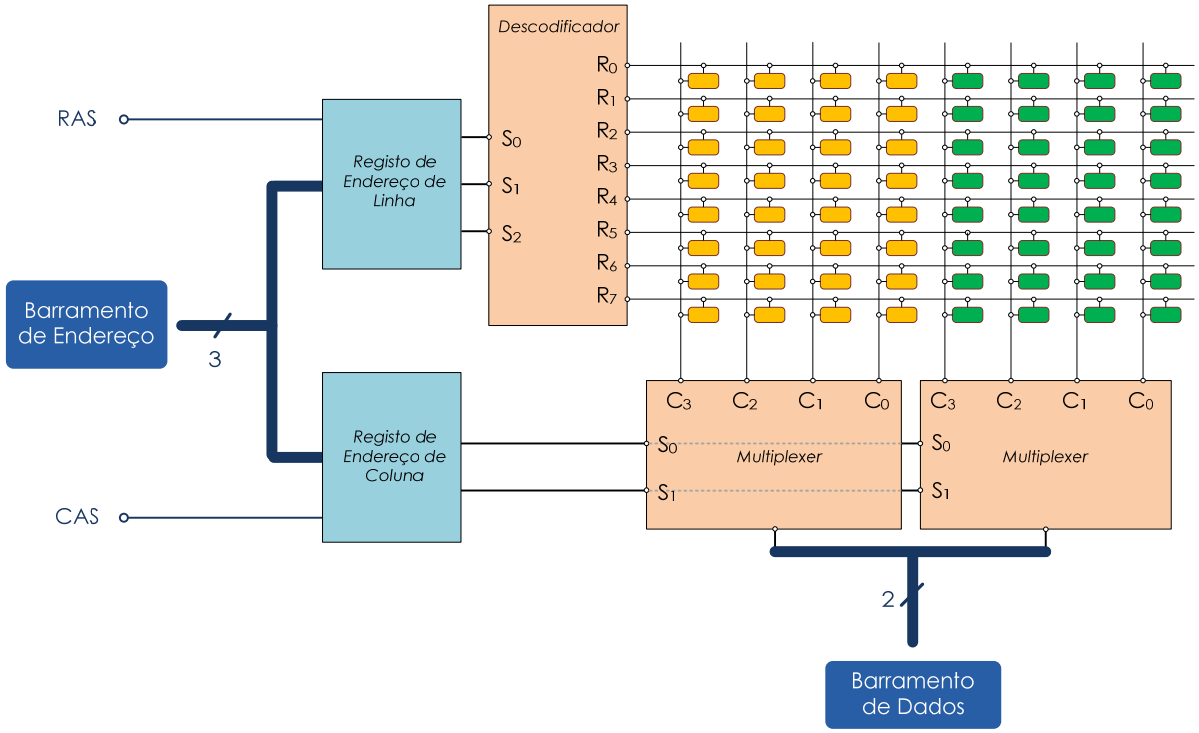


Figura 5.27 – Barramentos de endereço e de dados de uma memória.

O barramento de dados tem em geral um número de linhas igual ao número de bits de cada palavra. Mesmo que sejam armazenadas várias palavras numa mesma linha, só uma é colocada no barramento de dados de cada vez.

5.2.3 Plano de Memória

As aplicações e muitos dos dados utilizados no processamento são armazenados de forma permanente num dispositivo de memória não volátil como um disco rígido ou DVD. Esses dispositivos são muito lentos quando comparados com o processador e com as memórias DRAM. É portanto de todo o interesse que o código de uma aplicação caiba todo na memória principal para que a sua execução possa ser a mais rápida possível.

Em computadores pessoais é indispensável, hoje em dia, que seja possível ter várias aplicações a correr ao mesmo tempo. Quanto maior for a capacidade da memória principal do computador mais aplicações podem estar a correr ao mesmo tempo sem terem que ser periodicamente retiradas da memória principal e escritas na memória secundária (disco rígido).

É comum encontrar actualmente computadores com uma memória principal com capacidade entre 1 GB e 4 GB. Só existem, no entanto, circuitos integrados contendo memórias DRAM, com capacidades de algumas centenas de Mbit devido à tecnologia e processos de fabrico utilizados. De forma a construir um computador com mais memória do que essa, é necessário associar vários circuitos integrados, existindo várias formas de o fazer.

Em primeiro lugar pode ser necessário associar vários dispositivos de memória de forma a aumentar o tamanho de palavra. É possível ligar, por exemplo, 4 dispositivos de memória com tamanhos de palavra de 16 bits de forma a construir uma memória com palavras de 64 bits. Todos esses dispositivos recebem os mesmos sinais de controlo e endereço sendo o barramento de dados constituído pela junção dos barramentos de dados dos vários dispositivos. A um conjunto de dispositivos de memória ligados dessa forma dá-se o nome, em inglês, de *rank*.

Por outro lado, de forma a aumentar o número de palavras da memória, devem se ligar vários dispositivos de memória, ou *ranks*, que partilham as mesmas linhas de dados e cujo espaço de endereçamento é dividido, normalmente em blocos de igual tamanho.

No exemplo da Figura 5.28 tem-se uma memória com um tamanho de palavras de 64 bits construída a partir de dois *ranks* com 4 dispositivos de memória que possuem 16 bits de tamanho de palavra. O número de palavras desta memória é o dobro (dois *ranks*) do número de palavras existente em cada dispositivo de memória.

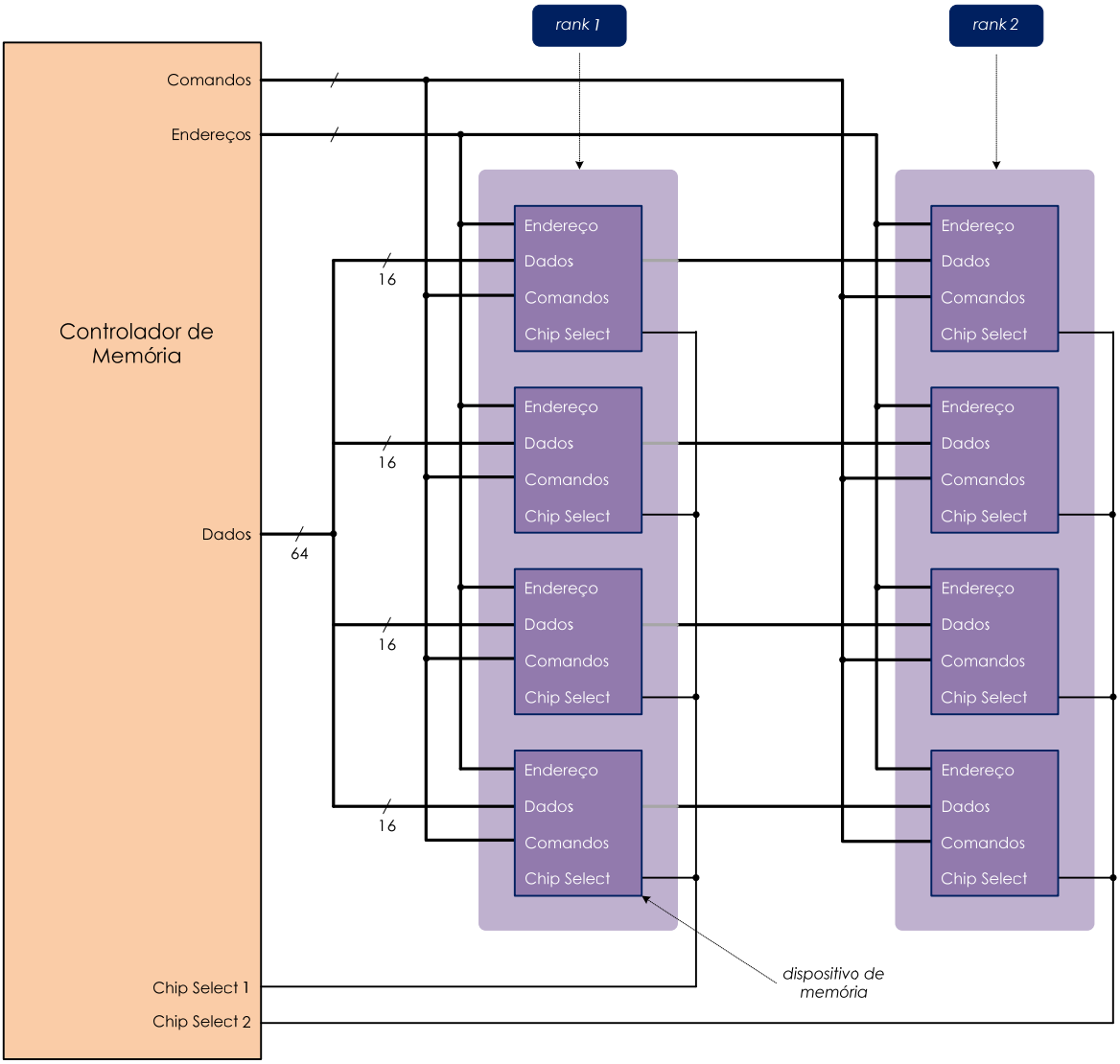


Figura 5.28 – Exemplo de uma memória constituída por dois *ranks* com 4 dispositivos de memória cada.

A organização da memória de um computador em partes independentes, como é o caso dos bancos de memória dentro dos circuitos integrados e dos *ranks*, permite uma maior rapidez de operação através da utilização de intercalagem. A intercalagem consiste em dividir o espaço de memória de forma a que palavras consecutivas sejam armazenadas em bancos (ou *ranks*) diferentes. Isso aumenta a largura de banda pois essas palavras podem ser lidas ou escritas em simultâneo.

A Figura 5.29 mostra um exemplo em que não é usada intercalagem. São usados dois bancos com a capacidade de *N* palavras cada dando origem a uma memória com 2*N* palavras. O espaço de memória é dividido a meio sendo a primeira metade armazenada no banco 0 e a segunda no banco 1. Neste caso o acesso consecutivo das palavras nos endereços 0 e 1, por exemplo, tem que ser feita uma depois da outra, isto é, só quando a palavra no endereço 0 tiver sido lida é que se pode ler a palavra no endereço 1.

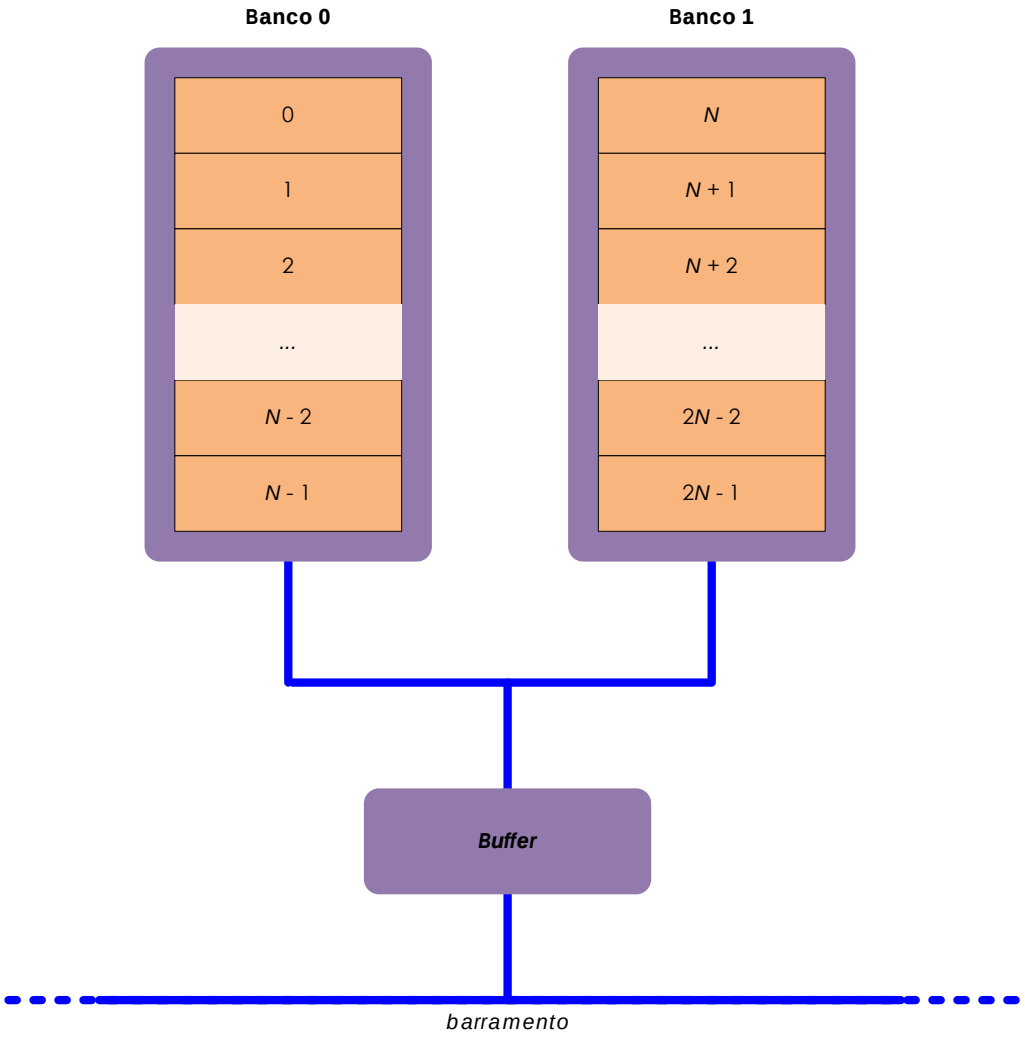


Figura 5.29 – Exemplo de uma organização de memória constituída por dois bancos sem a utilização de intercalagem.

A Figura 5.30, por outro lado, mostra uma memória com a mesma capacidade constituída por bancos com o mesmo tamanho mas em que as posições de memória alternam entre um e outro banco. O endereço 0 é armazenado no banco 0, o endereço 1 no banco 1, o endereço 2 no banco 0 e assim sucessivamente. Um acesso consecutivo aos endereços 0 e 1, por exemplo, é duas vezes mais rápido pois a leitura da informação da memória pode ser feita simultaneamente.

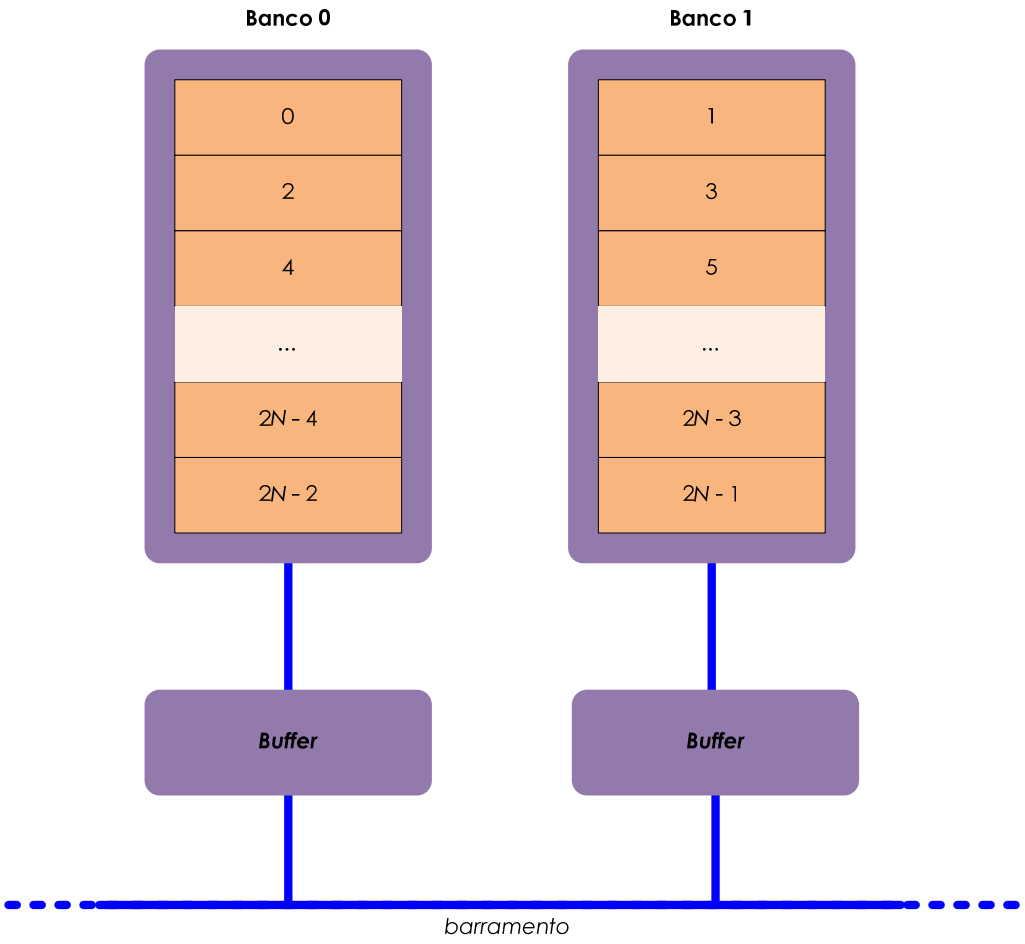


Figura 5.30 – Exemplo de uma organização de memória constituída por dois bancos em que é usada intercalagem.

No exemplo dado, a velocidade do barramento pode ser o dobro quando é usada intercalagem.

5.2.4 Exemplo

Imagine-se que se pretende projectar um sistema de memória dinâmica baseado em circuitos integrados do tipo MT16D232 numa organização com um único plano de memória e que o sistema deve dispor de um total de 32 MB, organizados em palavras de 32 bits. A Figura 5.31 mostra um extracto da folha de especificação destes módulos de memória.

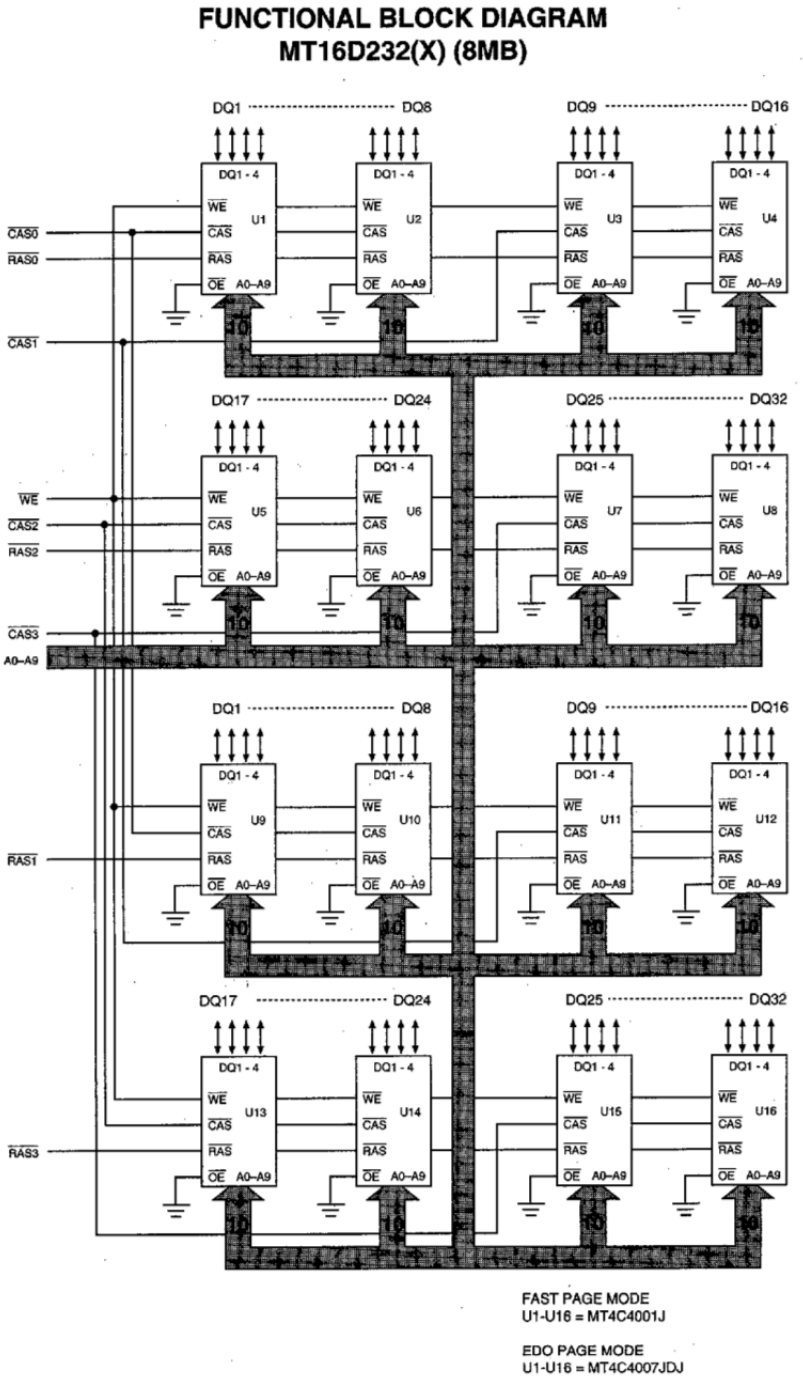


Figura 5.31 – Extracto da folha de especificações dos módulos de memória DRAM da Micron, referência MT16D232, mostrando a sua organização interna.

Cada módulo to tipo SIMM (*Single In-line Memory Module*), como se pode ver na folha de especificações, possui 16 circuitos integrados de memória DRAM (MT4C4001J) com 1 M palavras com 4 bits de comprimento. Tem uma 1 M palavras porque têm 10 linhas de endereço que são usadas alternadamente para endereçar a linha e a coluna. Imagina-se então uma disposição quadrada para

grupos de 4 células de memória – 1024 (10^{10}) linhas por 1024 colunas. Estimam-se que sejam grupos de 4 células de memória pois cada DRAM tem 4 linhas de dados para o exterior.

Quanto ao controlo da memória, embora haja uma linha de RAS e uma de CAS para cada circuito integrado DRAM (16 RAS e 16 CAS o todo), para o exterior do módulo SIMM só há 4 linhas de RAS e 4 de CAS. Isto significa que essas linhas que são acessíveis do exterior controlam simultaneamente mais do que uma DRAM (4 circuitos integrados de cada vez, neste caso). As 4 linhas RAS, por exemplo comandam ao mesmo tempo as 4 DRAMs de cada fila no diagrama de blocos apresentado na folha de especificações: 1 linha RAS para U1, U2, U3 e U4 (RAS0), outra para U5, U6, U7 e U8 (RAS2), outra para U9, U10, U11 e U12 (RAS1) e outra para U13, U14, U15 e U16 (RAS3). Da mesma forma as 4 linhas CAS comandam simultaneamente as DRAM U1, U2, U9 e U10 (CAS0), U3, U4, U11 e U12 (CAS1), U5, U6, U13 e U14 (CAS2), U7, U8, U15 e U16 (CAS3).

Estas combinações permitem usar a mesma SIMM para construir diferentes planos de memória. No caso do presente exemplo pretendem-se palavras de 32 bits. Há portanto que usar 8 DRAMs de cada SIMM para construir uma palavra. Como a SIMM tem 16 DRAMs, é possível ter 2 palavras de 32 bits para cada valor de endereço da SIMM (linhas A0-A9). A selecção entre uma ou outra palavra é feita activando simultaneamente as linhas de comando RAS0 e RAS2 (para uma palavra) ou RAS1 e RAS3 (para a outra palavra que partilha o mesmo endereço).

Cada SIMM pode ser encarada, portanto, como tendo 2 mega palavras de 32 bits. Para se construir o plano de memória desejado com 32 MB, ou seja 8 mega palavras de 32 bits (8 M x 4 bytes = 32 MB), são precisas 4 SIMMs.

As 4 SIMMs encaradas como tendo duas metades cada (metade de cima e metade de baixo do diagrama de blocos da SIMM) perfazem 8 blocos que são activados através das seguintes linhas de controlo:

- RAS0/RAS2 da SIMM1
- RAS1/RAS3 da SIMM1
- RAS0/RAS2 da SIMM2
- RAS1/RAS3 da SIMM2
- RAS0/RAS2 da SIMM3
- RAS1/RAS3 da SIMM3
- RAS0/RAS2 da SIMM4
- RAS1/RAS3 da SIMM4

Estas linhas de controlo são activadas uma de cada vez de acordo com os endereços da memória total. Como o plano de memória tem de ter 8 M palavras de 32 bits cada, são necessárias 23 linhas para endereçar essas palavras (2^{23} = 8 M palavras). Admitindo que não se usa intercalagem serão usados os 3 bits mais significativos da palavra de endereço (A20 a A23) para, através de um descodificador 3:8, activar as 8 linhas de RAS dos módulos SIMMs. Esse descodificador recebe na sua entrada de habilitação (*enable*) o sinal RAS gerado pelo controlador de memória. Assim, enquanto o

sinal RAS estivar desactivado (nível alto) nenhuma das saídas está activada (nível alto). Quando o sinal RAS for activado então uma das saídas, consoante os bits de endereço, será activada.

As restantes 20 linhas de endereço são multiplexadas em 2 conjuntos de 10 linhas para fornecimento às SIMMs (e DRAMs nelas contidas) para indicação da linha e coluna a activar. Esse multiplexer é controlado pelo sinal geral de CAS. Relembre-se que o protocolo de acesso à memória DRAM consiste em:

1) colocar no barramento de endereços o endereço da linha,
2) activar o sinal RAS,
3) colocar no barramento de endereços o endereço da coluna,
4) activar o sinal CAS depois de passado um certo tempo mínimo do flanco descendente do sinal RAS (*RAS-to-CAS delay time*: t_{RCD}).

Assim sendo, quando o endereço de linha está no barramento de endereços, o sinal CAS está no nível alto e portanto o multiplexer coloca na sua saída as linhas de endereço A10 a A19 que são usadas para indicar a linha. Quando o sinal CAS é activado (nível baixo) o multiplexer coloca na sua saída as linhas de endereço A0 a A9 que são usadas para seleccionar a coluna. Note-se que esta escolha não podia ser ao contrário porque palavras de memória pertencentes a endereços consecutivos devem estar armazenadas na mesma linha de modo a por ser usar o modo rápido de página em que um conjunto de palavras, chamadas de "página", podem ser acedidas activando unicamente a linha CAS com o endereço de coluna apropriado no barramento de endereços sem ter que se activar e desactivar a linha de RAS entre cada acesso, o que torna o procedimento mais lento devido à necessidade de pré-carga e equalização das linhas.

Note-se ainda que em algumas memórias os endereços devem estar no barramento um certo tempo antes das linhas de RAS ou CAS serem activadas. Assim sendo, e como no plano de memória proposto os endereços de coluna só estão na linha depois de o multiplexer os seleccionar, o que só é feito depois do sinal CAS ser activado, é preciso introduzir um atraso no sinal CAS entre a entrada do multiplexer e a entrada das SIMMs de modo a que a transição acabe por acontecer, na perspectiva da memória DRAM, depois dos endereços estarem no barramento. Isso pode ser feito introduzindo no caminho do sinal CAS duas ou mais (em número par) portas lógicas NOT.

As linhas de controlo CAS e WE são também ligadas em paralelo a todos os módulos SIMM já que a sua activação numa dada memória DRAM quando a linha RAS está inactiva não tem qualquer influência. Assim sendo só a DRAM cuja linha RAS foi activada (por intermédio do descodificador 3:8 usado) é que prestará atenção ao nível desses sinais de controlo (CAS e WE).

O controlador de memória é responsável por gerar os sinais RAS e CAS para as SIMMs com a temporização adequada. A interface que apresenta para o exterior, isto é, para o CPU, consiste nas linhas de relógio, de leitura/escrita, de selecção do dispositivo (*chip select*) e linha de READY para indicação de quando a operação está concluída. Essa interface é igual quer se trata de uma memória SRAM ou DRAM. Esse aspecto é transparente para o processador bem como o facto de existir ou não memória tampão (*cache*), por exemplo.

A Figura 5.32 apresenta então a organização do plano de memória contendo os módulos de memória, o controlador, o descodificador para as linhas RAS e o multiplexer para selecção das linhas de endereço enviadas para os módulos de memória.

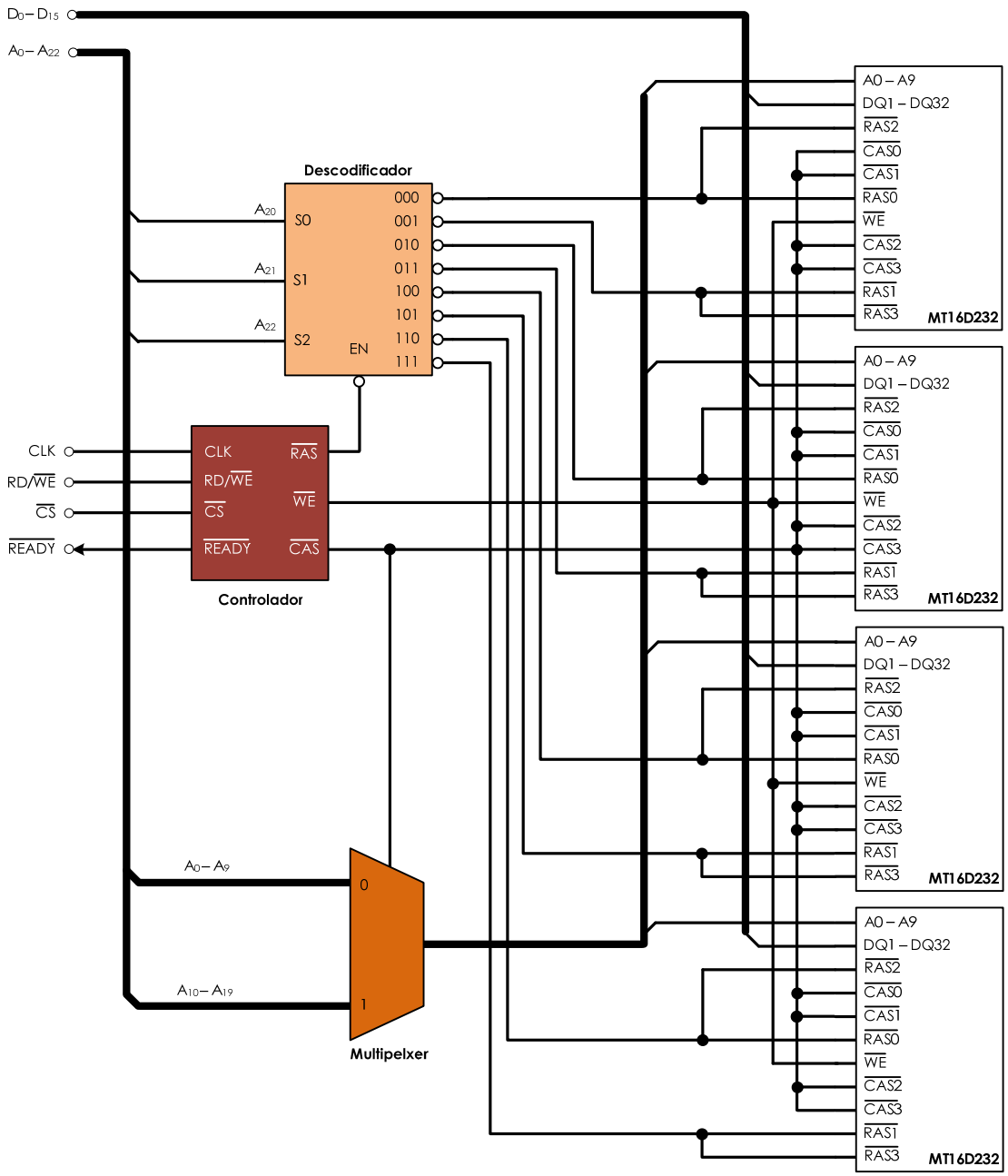


Figura 5.32 – Organização do plano de memória do exemplo.

5.3 Memória SRAM

5.3.1 Célula de Memória SRAM

Como se viu anteriormente, na memória SRAM é usado um circuito bi-estável (*flip-flop*) para armazenar a informação (Figura 5.23). O armazenamento de dados é volátil, isto é, se for desliga a alimentação a informação é perdida. No entanto, ao contrário da célula de memória DRAM, a

informação não desaparece com o tempo. Nas memória DRAM a informação é aramzenda como carga eléctrica num condensador. Essa carga vai, com o tempo, escoando para o meio evolvente.

A ligação desse circuito à matriz de endereçamento é feita usados dois transístores de acesso (Q5 e Q6) controlados pela mesma linha de palavra, como ilustrado na Figura 5.33.

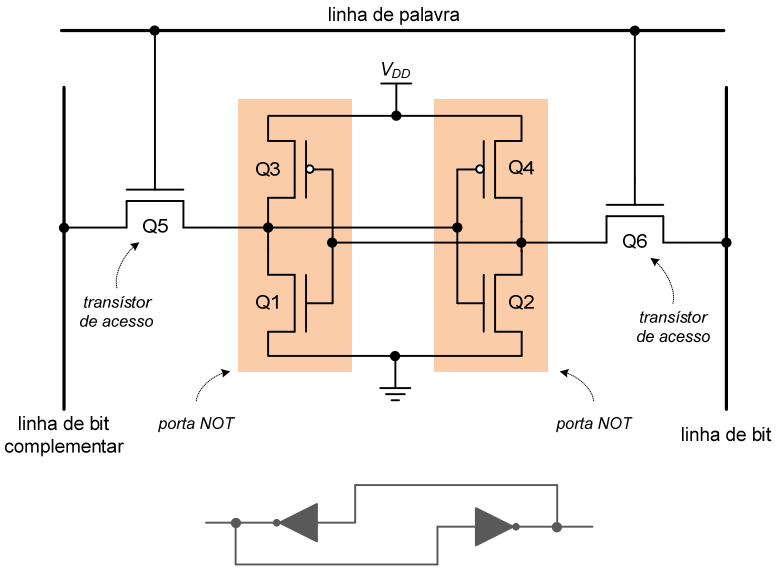


Figura 5.33 – Célula de memória SRAM (tipo 6T).

Note-se que são usadas duas linhas de dados complementares para transferência do bit armazenado de e para o exterior**Erro! A origem da referência não foi encontrada.**. O dimensionamento do tamanho dos transístores MOSFET usados é crítico para que se consiga ter a menor área e consumo possível e ao mesmo tempo manter a informação armazenada sem erros o mais tempo possível, ser capaz de ler essa informação de forma rápida sem a destruir e escrever nova informação de forma a garantir que fica armazenada na célula.

A Figura 5.34 mostra a organização de uma memória SRAM onde as duas linhas de bit complementares são ligadas a um amplificador antes de chegarem ao multiplexer de selecção de coluna.

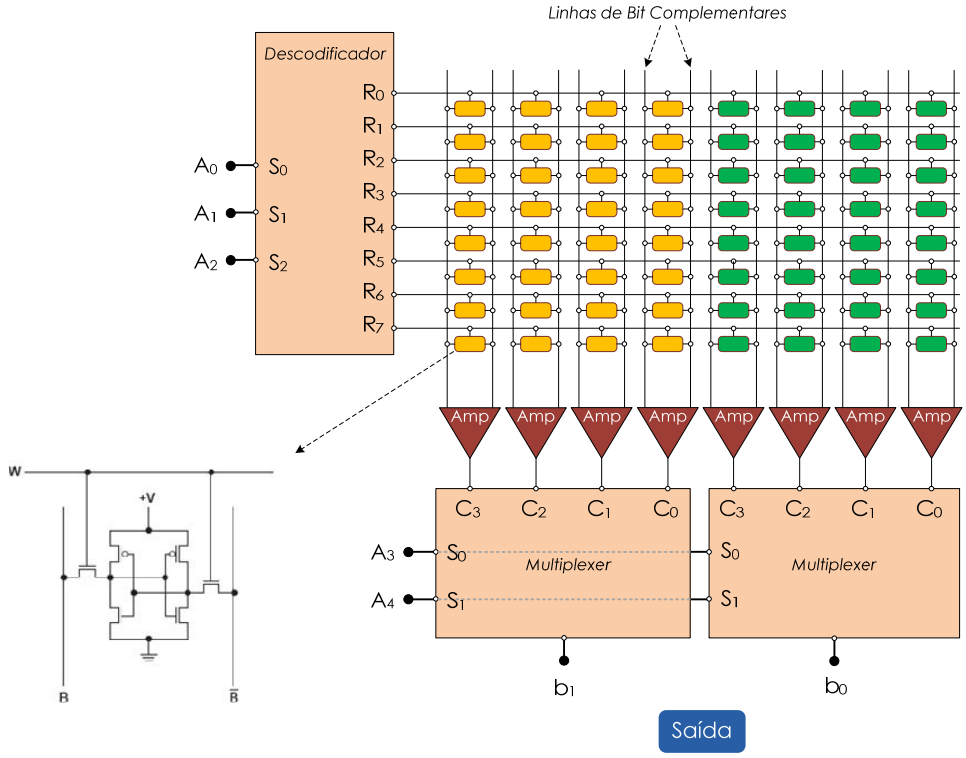


Figura 5.34 – Exemplo da organização matricial das células de memória numa memória SRAM de 64 bits com palavras de 2 bits.

5.3.2 Tecnologias de Célula

A célula de memória SRAM apresentada na Figura 5.33 é usualmente conhecida por "6T" por usar 6 transistores CMOS (4 NMOS e 3 PMOS). De forma a melhorar o desempenho desta célula diferentes variantes têm sido propostas e usadas que têm, no entanto, algumas desvantagens. No tipo "4T", apresentado na Figura 5.35, são usados só 4 transístores. Os 2 transistores PMOS foram substituídos por resistências. Essas resistências são implementadas numa camada de sílicio diferente feita com polisilício de alta resistividade por forma a implementar as resitências, da ordem dos GΩ, na menor área possível. Embora sejam menores do que as células do tipo 6T são ainda assim maiores dos que as células das memórias DRAM. Essas resitências têm de ter um valor elevado de forma a minimizar a corrente usada e portanto o consumo energético da célula. Isso leva, no entanto a que as estas células sejam mais susceptíveis ao ruído e que tenha um tempo de acesso maior.

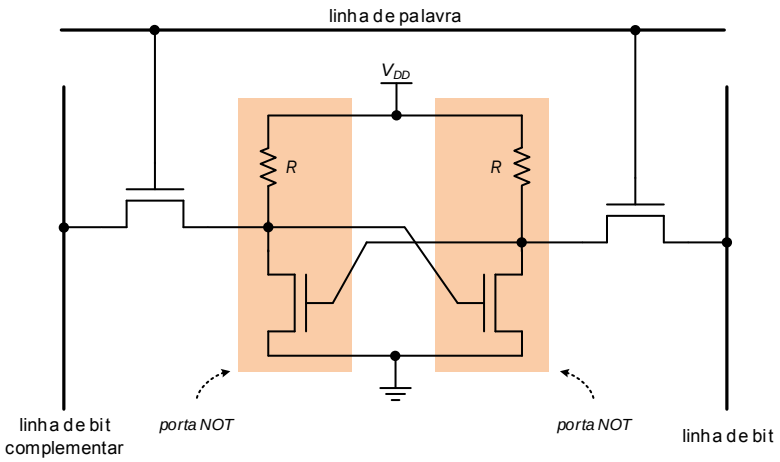


Figura 5.35 – Célula de memória SRAM do tipo 4T.

5.3.3 Leitura

Em memórias pequenas, como as encontradas nos registos dos processadores até em memórias tampão, a saída da célula de memória pode ser usada directamente como indicação do valor do bit armazenado. Em memórias maiores, no entanto, devido ao elevado número de células por coluna, o tamanho das linhas de bit é muito grande o que faz com que a sua capacidade seja elevada. Isso, por seu lado, leva a que, depois de ligada a célula de memória à linha de bit, a tensão dessa linha demore algum tempo a atingir o valor imposto pela célula. Para tornar esse processo mais rápido é comum encontrar, em memórias maiores, amplificadores diferenciais ligados às duas linhas de bit complementares de uma coluna de modo a tornar mais rápida a determinação do valor do bit armazenado numa célula.

A Figura 5.36 mostra o esquema eléctrico de um amplificador que é constituído por duas portas NOT cruzadas, tal como na célula de memória. Este amplificador é no entanto construído com transístores maiores e que permitem uma maior corrente e portanto uma maior rapidez de carga e descarga das capacidades das linhas de bit.

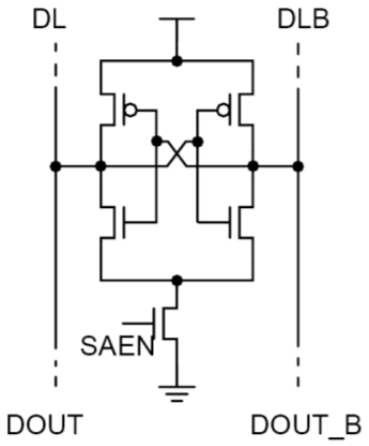


Figura 5.36 – Amplificador de detecção na tensão nas linhas de bit.

Note-se que este amplificador é accionado através do sinal SAEN que só é activado algum tempo depois de a célula de memória ter sido ligada às linhas de bit.

De forma a diminuir o número de amplificadores necessários em memórias que tenham muitas colunas, é comum o uso de multiplexers antes dos amplificadores. Esses multiplexers são constituídos por dois transístores pMOS para cada coluna controlados por um sinal proveniente de um descodificador que recebe o endereço de coluna. Como ilustra a Figura 5.37 várias colunas são assim ligadas a um só amplificador.

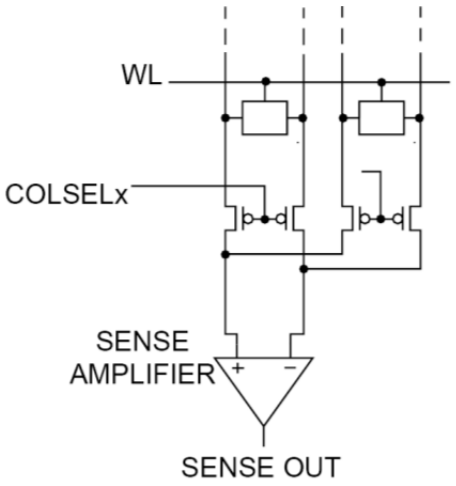


Figura 5.37 – Circuito de multiplexagem das colunas para leitura dos bits.

Em dispositivos de memória SRAM externos, com capacidade de vários megabit e em que o tamanho das palavras é geralmente de 8 bits, usam-se vários níveis de amplificação e multiplexagem, como ilustra a Figura 5.38 para o caso de 3 níveis.

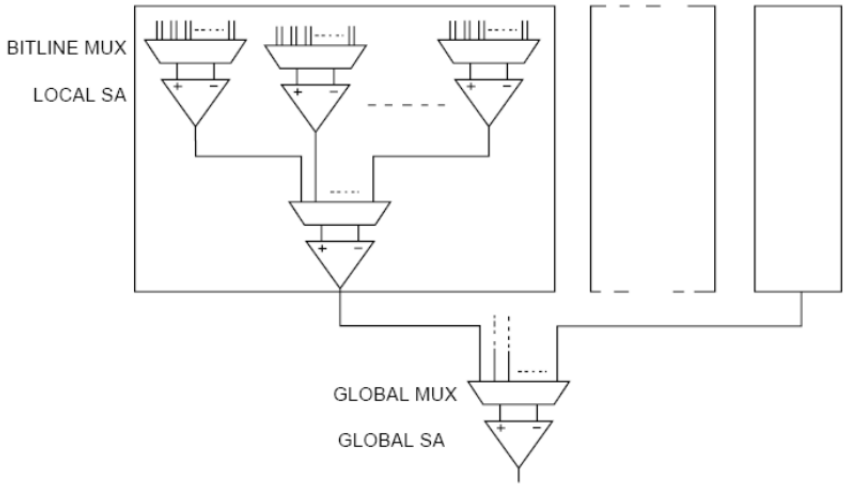


Figura 5.38 – Ilustração do uso de vários níveis de amplificação e multiplexagem numa memória SRAM.

O valor de saída do amplificador de leitura é conectado a um registo cuja função é armazenar o valor do bit lido para ser disponibilizado para o exterior enquanto os circuitos internos na memória podem começar a efectuar a leitura ou escrita seguinte.

5.3.4 Pré-carga e Equalização

Antes mesmo de se conectar as células de memória às linhas de bit, estas são colocadas a uma tensão pré-definida, em geral V_{DD}. Essa operação é também designada de pré-carga pois implica a transferência de carga eléctrica para as linhas de bit (que se comportam como condensadores). Depois de as células de memória se ligarem às linhas de bit, uma delas (de cada coluna) será "puxada" para 0 pela célula enquanto a outra manter-se-á no nível alto.

A Figura 5.39 mostra a evolução temporal da tensão das linhas de bit (BL e BLB) e da saída do amplificador (SENSE OUT) depois de activada a linha de palavra (WL) e seleccionada a coluna (COLSELx).

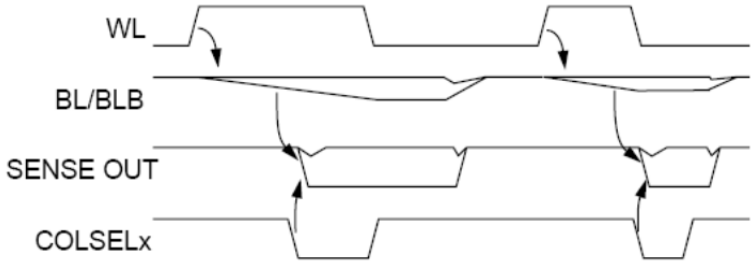


Figura 5.39 – Evolução temporal da tensão das linhas de bit (BL e BLB) e da saída do amplificador (SENSE OUT) depois de activada a linha de palavra (WL).

Como foi referido, é necessário que as linhas de bit sejam pré-carregadas. O valor de tensão em ambas as linhas deve inicialmente ser exactamente o mesmo para que o amplificador possa operar correctamente. A Figura 5.40 mostra o esquema eléctrico de um circuito tipicamente utilizado para a

pré-carga e equalização. É constituído por 3 transistores pMOS que quando activados pelo sinal comum de pré-carga (PRE#) conectam as linhas de bit a V_{DD} e entre si.

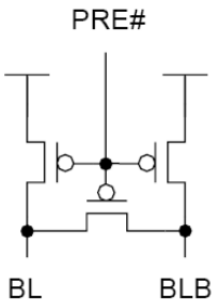


Figura 5.40 – Circuito de pré-carga e equalização.

5.3.5 Escrita

No caso da escrita de um valor na célula de memória é necessário forçar as linhas de bit complementares da coluna desejada a um nível de tensão alto e baixo. Isso é feito com dois transístores nMOS ligados a 0 e controlados pela linha de dados externa. Como as linhas de bit são pré-carregadas no nível alto é só preciso que um dos transístores de escrita seja activado para a escrita de um bit.

Os transístores de escrita têm que ser maiores do que os transístores das células de memória para serem capazes de os contrariar quando se pretende mudar o valor do bit armazenado na célula. É usado também um multiplexer para ligar os amplificadores de escrita á coluna onde se encontra a célula em que se deseja escrever. A linha de dados é ligada a duas portas NOT que funcionam como buffers e que criam duas tensões complementares para controlo dos transístores de escrita conectados às linhas de bit complementares de uma coluna.

A Figura 5.41 mostra o conjunto dos circuitos usados para ler e escrever numa célula de memória do tipo SRAM.

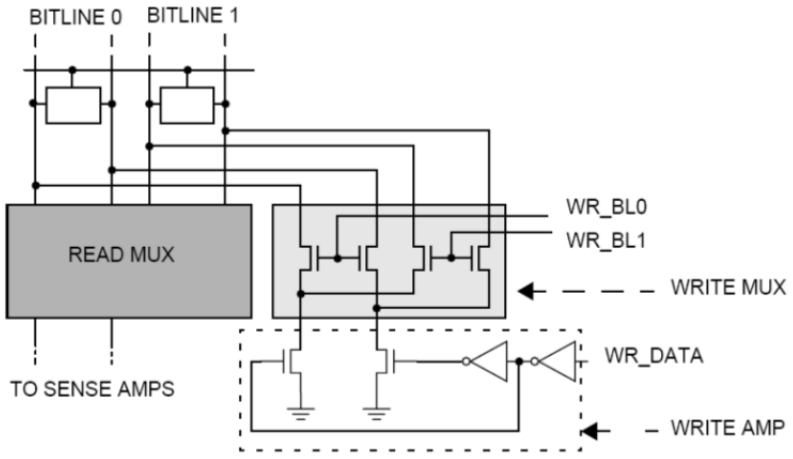


Figura 5.41 – Circuitos de leitura e escrita nas células de memória SRAM.

5.3.6 Resumo

Resumindo, a leitura de um valor de um bit (ou conjunto de bits) de uma memória SRAM consiste nos seguintes passos:

1. Pré-carregar as linhas de bit com V_{DD}.
2. Seleccionar a linha para que a célula de memória seja ligada às linhas de bit.
3. Ligar o amplificador de leitura à coluna adequada.
4. Activar o amplificador de leitura.
5. Activar o registo de saída.

No caso da escrita os passos são os seguintes:

1. Pré-carregar as linhas de bit com V_{DD}.
2. Seleccionar a linha adequada o que faz com que as células de memória sejam ligadas às linhas de bit.
3. Ligar o amplificador de escrita à coluna adequada.

A quantidade de carga que é preciso colocar nas linhas de bit depois de uma escrita é maior do que depois de uma leitura já que aí as linhas de bit nunca chegam a atingir a tensão 0. Por essa razão são usados dois circuitos de pré-carga e equalização das linhas de bit: um para depois da leitura e outro para depois da escrita. O circuito de pré-carga usado depois das escritas é maior e por isso mesmo consome mais energia, razão pela qual não é usado também depois das leituras.

Como as linhas ligadas à entrada do amplificador de leitura só estão conectadas às linhas de bit das colunas depois do multiplexer de leitura ter sido activado, é necessário pré-carregar e equalizar essas linhas ao mesmo tempo que se pré-carrega e equaliza as linhas de bit das colunas.

O circuito completo para uma coluna numa memória SRAM é apresentado na Figura 5.42.

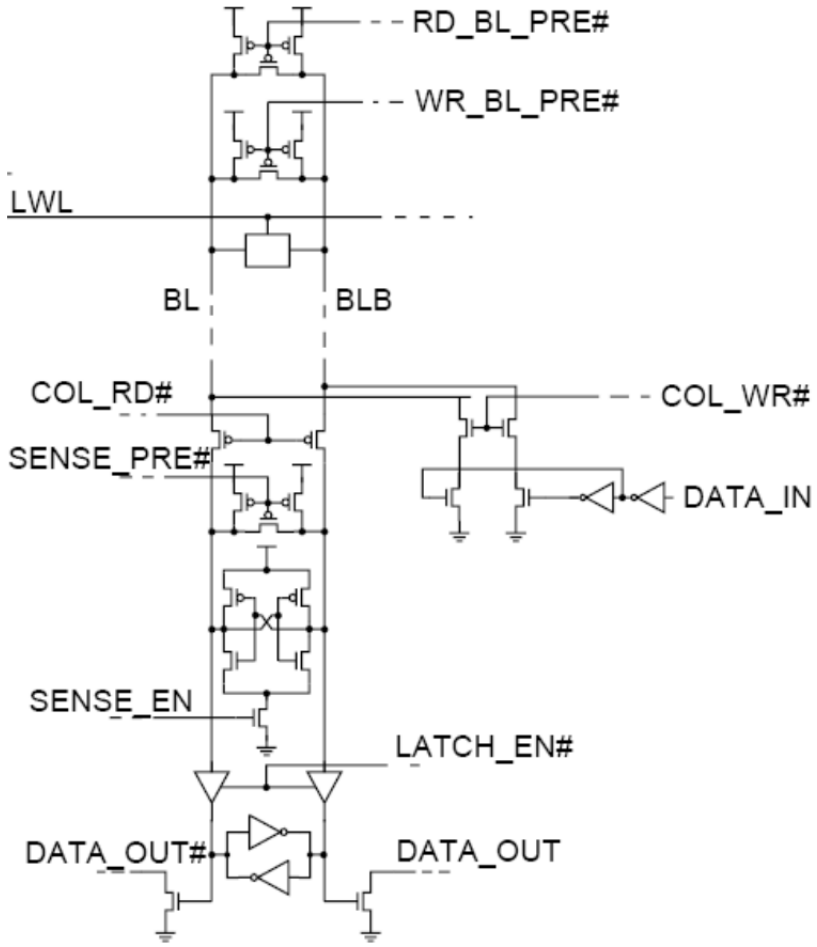


Figura 5.42 – Circuito eléctrico usado numa coluna de uma memória SRAM.

5.3.7 Sinais de Controlo

O controlo do acesso à memória SRAM é feito através das linhas de leitura/escrita (WE*), activação da saída (OE*) e selecção do dispositivo (CS*). A leitura e escrita da informação são desencadeadas pelo flanco negativo da linha CS. O valor da linha WE, nessa altura determina, se é para ler ou escrever os dados na memória (valor alto significa leitura).

A Figura 5.43 mostra o diagrama temporal de um sequência de operações constituída por uma leitura seguida de uma escrita.

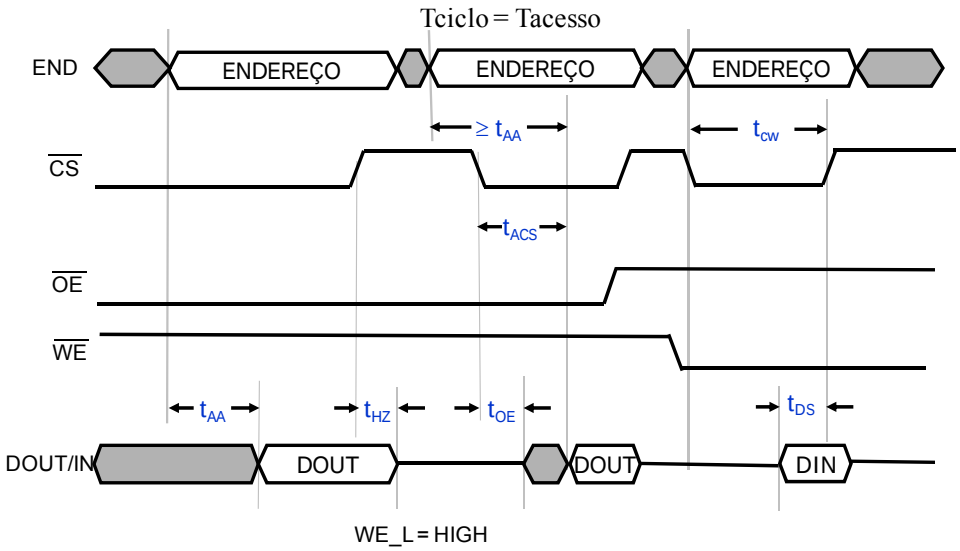


Figura 5.43 – Diagrama temporal dos sinais de controlo da memória SRAM.

As diferentes temporizações de interesse numa memória deste tipo são:

- t_{AA} (access time for address) – tempo para obter os dados na saída após depois de colocar novo endereço.
- t_{ACS} (access time for chip select) – tempo para obter os dados na saída após depois de activar o sinal CS.
- t_{OE} (output enable time) – tempo necessário para os bufferes de 3 estados passarem do estado de alta impedância quando os sinais de CS e OE são activados.
- t_{OZ} (output-disable time) – tempo necessário para os bufferes de 3 estados passarem ao estado de alta impedância quando os sinais de CS e OE são desactivados.
- t_{OH} (output-hold time) – tempo que os dados de saída se mantêm válidos após alterar o endereço.

5.4 Memória DRAM

5.4.1 Célula de Memória

Uma das diferenças em relação às memórias SRAM tem a ver com a célula de memória que neste caso é constituída por um condensador e um transístor MOS ligado entre este e a linha de bit e controlado pela tensão nas linhas de palavra (Figura 5.44).

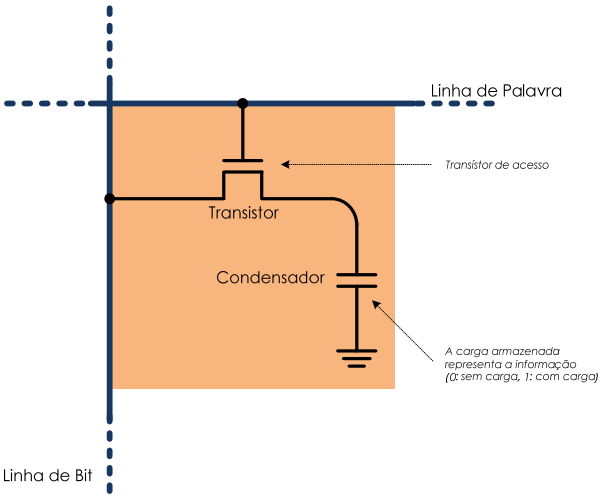


Figura 5.44 – Célula de memória DRAM.

A informação é representada pela carga eléctrica armazenada no condensador. Um bit 0 corresponde a um condensador sem carga e um bit 1 corresponde a um condensado com carga. A capacidade típica, usando-se um processo de fabrico de 90 nm, é da ordem dos 30 fF [10].

O transístor de acesso, quando a tensão de porta é 0, não tem uma resistência infinita como seria desejável. A corrente de fuga é tipicamente de 1 fA [10]. Isto faz com que o condensador só consiga manter um valor de carga suficiente para que não haja perda de informação durante algumas centenas de milissegundos. Por forma que em toda a memória não haja um único bit de informação que se perca, todas as células são refrescadas pelo menos uma vez em cada 32 ou 64 ms. Quanto mais pequenos foram os condensadores e transístores menor será a carga armazenada e maior será a corrente de fuga nos transístores de acesso o que dificulta muito o aumento de capacidade de memória através da miniaturização.

5.4.2 Leitura e Escrita

Os circuitos electrónicos existentes nas memórias DRAM são semelhantes aos das memórias SRAM. Em particular são usados também circuitos de pré-carga e equalização das linhas de bit, embora neste caso a tensão a que são colocadas essas linhas seja $V_{DD}/2$ e não V_{DD} como acontece tipicamente com as memórias SRAM. Da mesma forma são usados amplificadores de leitura e de escrita e multiplexers para a ligação desses amplificadores às colunas adequadas.

No caso das memórias SRAM cada célula está ligada a duas linhas de bit (complementares). No caso das células na memória DRAM só é necessária uma linha de bit, como se ilustra na Figura 5.44. Tendo em conta a organização matricial das célula de memória normalmente utilizada ter-se-ia uma organização como a ilustrada na Figura 5.45.

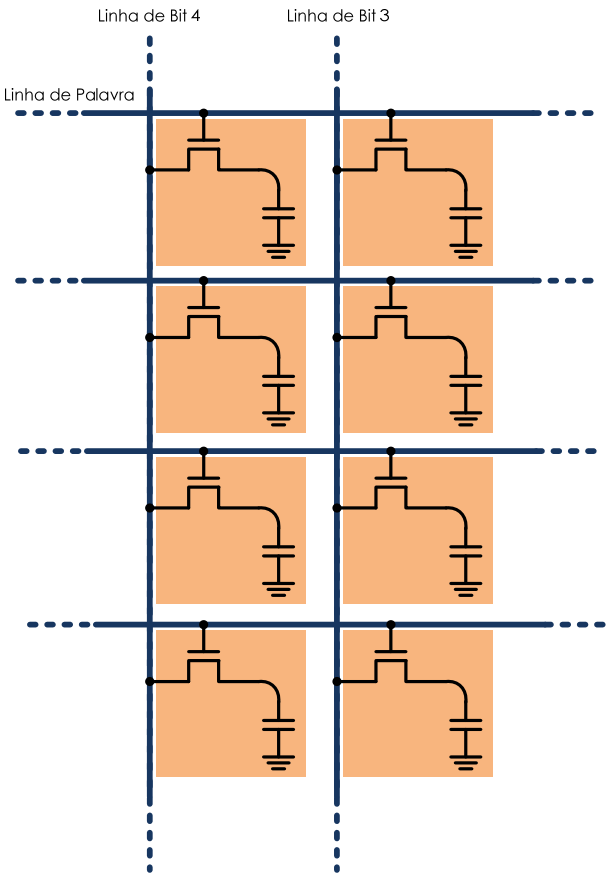


Figura 5.45 – Disposição de várias célula de memória DRAM.

Na verdade, para que a leitura seja feita da mesma forma do que nas memórias SRAM cada coluna, isto é, cada conjunto de células ligada a uma linha de bit é dividida em dois de modo que metade das células de memória seja ligada a uma linha de bit e a outra metade a uma outra linha de bit como se ilustra na Figura 5.46.

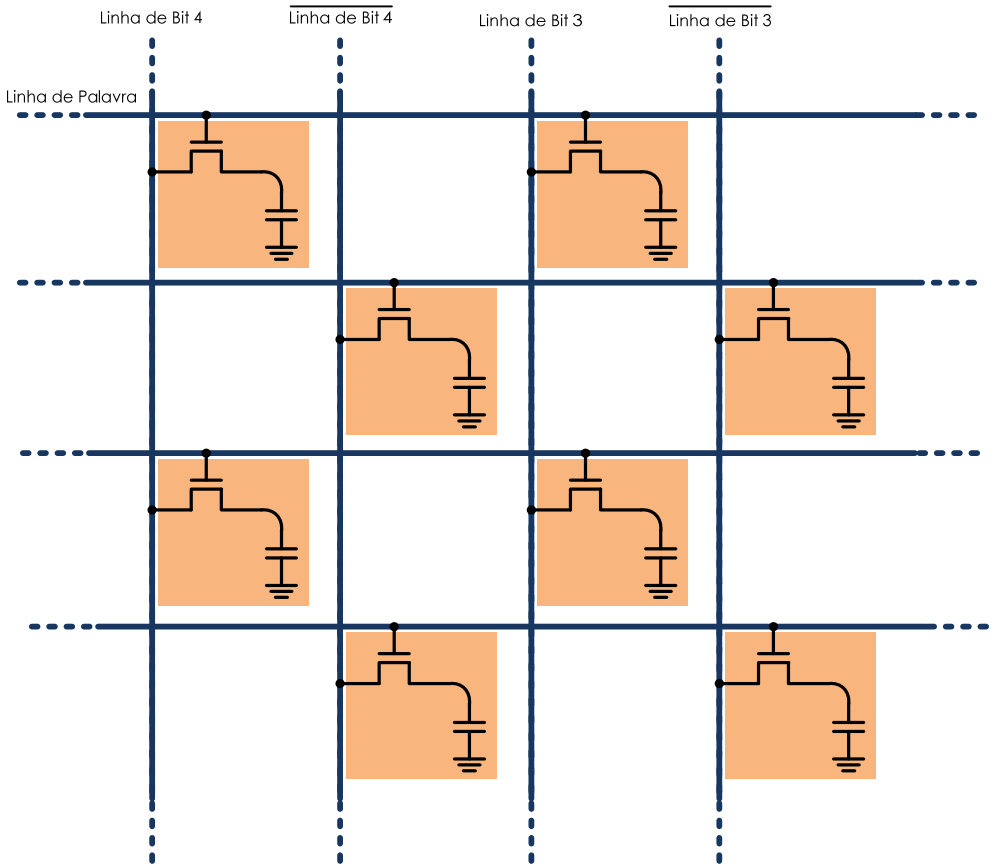


Figura 5.46 – Disposição de várias célula de memória DRAM usando duas linhas de bit por coluna.

No *layout* do circuito integrado onde este tipo de memória é implementada as duas linhas de bit de uma mesma coluna são "entrançadas" como ilustra a Figura 5.47. Isso faz com que a interferência electromagnética provocada por circuitos próximo afecte, tanto quando possível, as duas linhas da mesma forma.

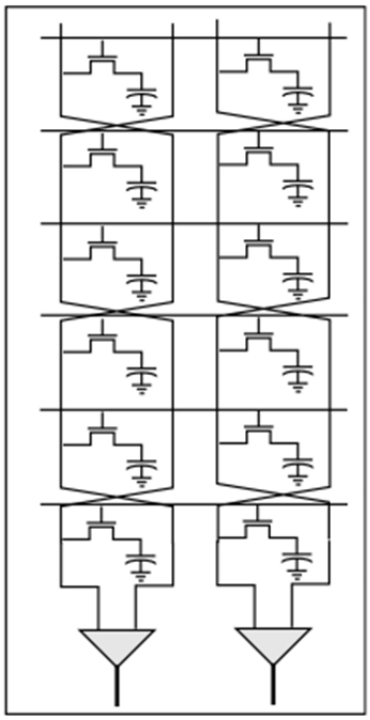


Figura 5.47 – Ilustração do entrelaçamento das linhas de bit de uma memória DRAM.

É usado um amplificador de leitura diferencial de modo a detectar o valor do bit guardado na célula de memória. Essa informação é armazenada na forma de carga eléctrica num condensador – a presença de carga representa um "1" e a ausência de carga representa um "0". Como as linhas de bit são em geral bastante longas, possuem uma capacidade significativa comportando-se efectivamente como um condensador com uma capacidade que é em geral superior à dos próprios condensadores usados para armazenar a informação. Assim sendo, quando se liga um desses condensadores à linha de bit, a sua carga, caso exista, irá passar para a linha de bit fazendo aumentar a tensão desta. Caso o bit guardado na célula seja "0" o condensador irá receber carga da linha de bit fazendo com que a tensão desta desça. É por esta razão que as linhas de bit são pré-carregadas a $V_{DD}/2$ e não a V_{DD} – para que a sua tensão possa subir ou descer em relação a esse nível de referência.

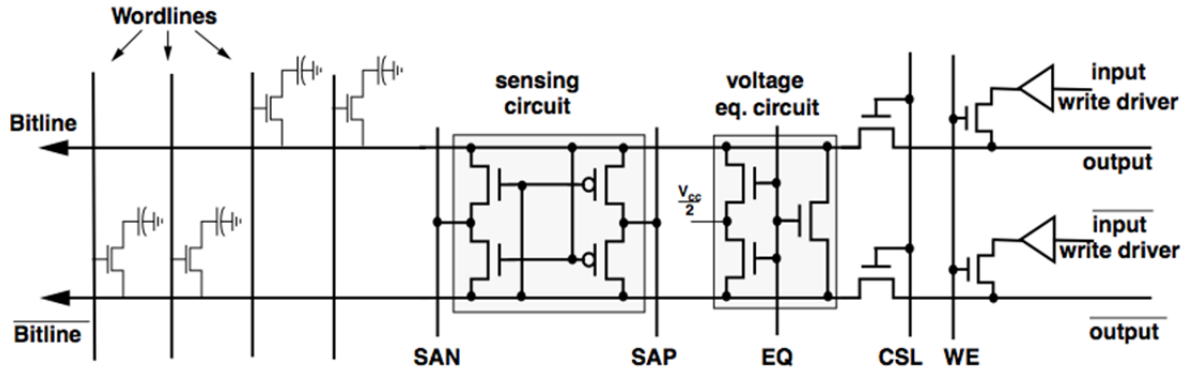


Figura 5.48 – Circuitos de leitura e escrita de numa célula de memória DRAM.

Essa transferência de carga é em geral lenta, tendo em conta os padrões de rapidez de acesso existentes hoje em dia, e depende do valor das capacidades do condensador e da linha de bit. O amplificador de leitura irá, consoante a variação sentida na tensão da linha, produzir à sua saída uma tensão baixa (0) ou alta (V_{DD}) de forma a representar a informação lida da memória. Esse amplificador é do tipo diferencial, tal como nas memórias SRAM. Uma das entradas é ligada à linha de bit da coluna desejada e a outra entrada é ligada à linha de bit de uma coluna adjacente servindo como valor de referência pois essa linha de bit (como todas as outras) é pré-carregada a $V_{DD}/2$. Como essa linha de bit da coluna adjacente não está, nesse instante, ligada a nenhuma célula de memória, a sua tensão manter-se-á em $V_{DD}/2$ até o amplificador de leitura ser activado.

O amplificador de leitura é constituído por 4 transístores que formam um circuito bistável, isto é, com dois pontos de funcionamento possíveis em condições estacionárias (Figura 5.49).

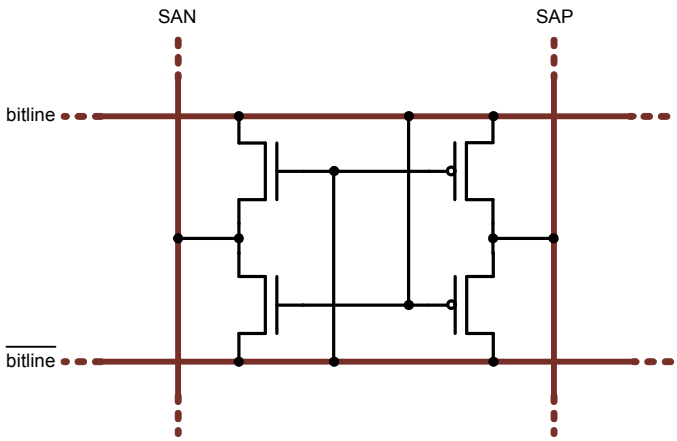


Figura 5.49 – Amplificador de leitura.

Este circuito bistável não são mais do que duas portas NOT (Figura 5.50). As suas alimentações positiva e negativa estão ligadas às linhas SAP e SAN respectivamente.

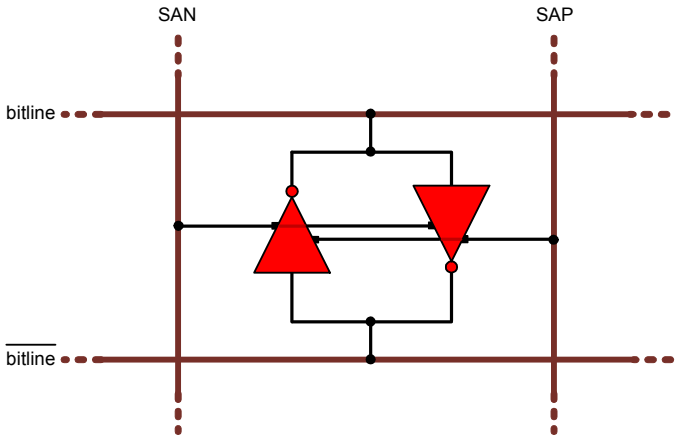


Figura 5.50 – Amplificador de leitura constituído por duas portas NOT.

A ligação de ambas as alimentações de uma porta NOT a $V_{DD}/2$ faz com que ambos os transístores não estejam a conduzir e portanto a saída da porta está livre de ter um qualquer valor de tensão (Figura 5.51). Nesta caso as saídas das duas portas NOT irão ter nas suas saídas o valor de tensão das linhas de bit a que estão ligadas.

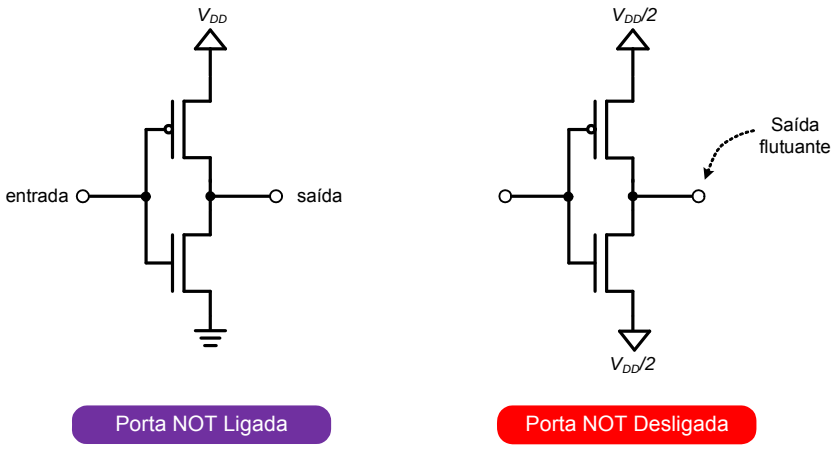


Figura 5.51 – Porta NOT ligada e desligada.

O amplificador de leitura tem uma função de transferência que resulta da combinação das funções de transferência das duas portas NOT.

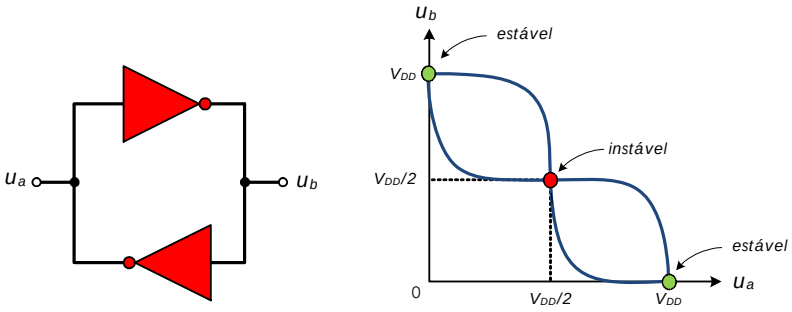


Figura 5.52 – Função de transferência do circuito bistável constituído por duas portas NOT.

Quando um condensador onde está armazenado o bit a ler é ligado à sua linha de bit a tensão dessa linha irá subir ou descer ligeiramente em relação ao valor de $V_{DD}/2$. A Figura 5.53 mostra o caso em que essa tensão desce um pouco (Δ) dando origem a um ponto de funcionamento estável em que a tensão da linha de bit é 0 e a da linha de bit complementar é V_{DD}.

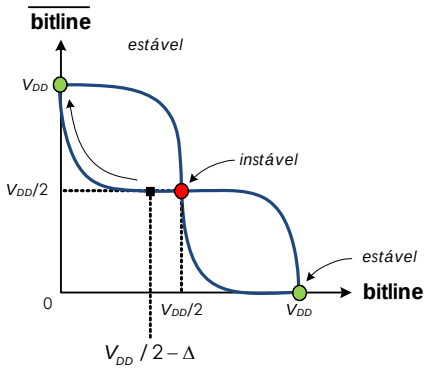


Figura 5.53 – Comportamento do amplificador de leitura quando a tensão na linha de bit descer um pouco em relação ao valor de referência ($V_{DD}/2$).

Para que o amplificador diferencial funcione correctamente na detecção da direcção em que variou a tensão da linha de bit (para cima ou para baixo), e para que o faça de forma rápida, é muito importante que inicialmente a tensão nas duas linhas de bit a que está conectado seja exactamente a mesma. Isso é conseguido por um lado com o circuito de pré-carga e equalização da Figura 5.40 .

Note-se que, ao contrário do que sucede na memória SRAM, no caso da memória DRAM o uso de amplificadores de leitura é obrigatório pois os condensadores das células de memória não têm capacidade, só por si, de alterarem substancialmente o valor da tensão das linhas de bit.

Uma outra diferença tem a ver com o facto de que quando se liga uma célula de memória a uma linha de bit, a informação que lá estava armazenada pode ser corrompida dado que o valor da carga armazenada altera-se. Isso, no entanto, não é problema já que o uso de um amplificador como o da Figura 5.36, em que não há distinção entre entrada e saída, faz com que à medida que a variação da tensão da linha de bit vai sendo amplificada o nível de carga dos condensadores vai sendo restabelecido. Por exemplo, no caso em que o bit armazenado vale 1 (condensador carregado) inicialmente a carga do condensador vai diminuir e a tensão da linha de bit vai aumentar ligeiramente. Logo que o amplificador de leitura é ligado a tensão dessa linha de bit vai aumentar bastante devido à sua acção o que provoca o recarregamento do condensador que se mantém conectado à linha de bit.

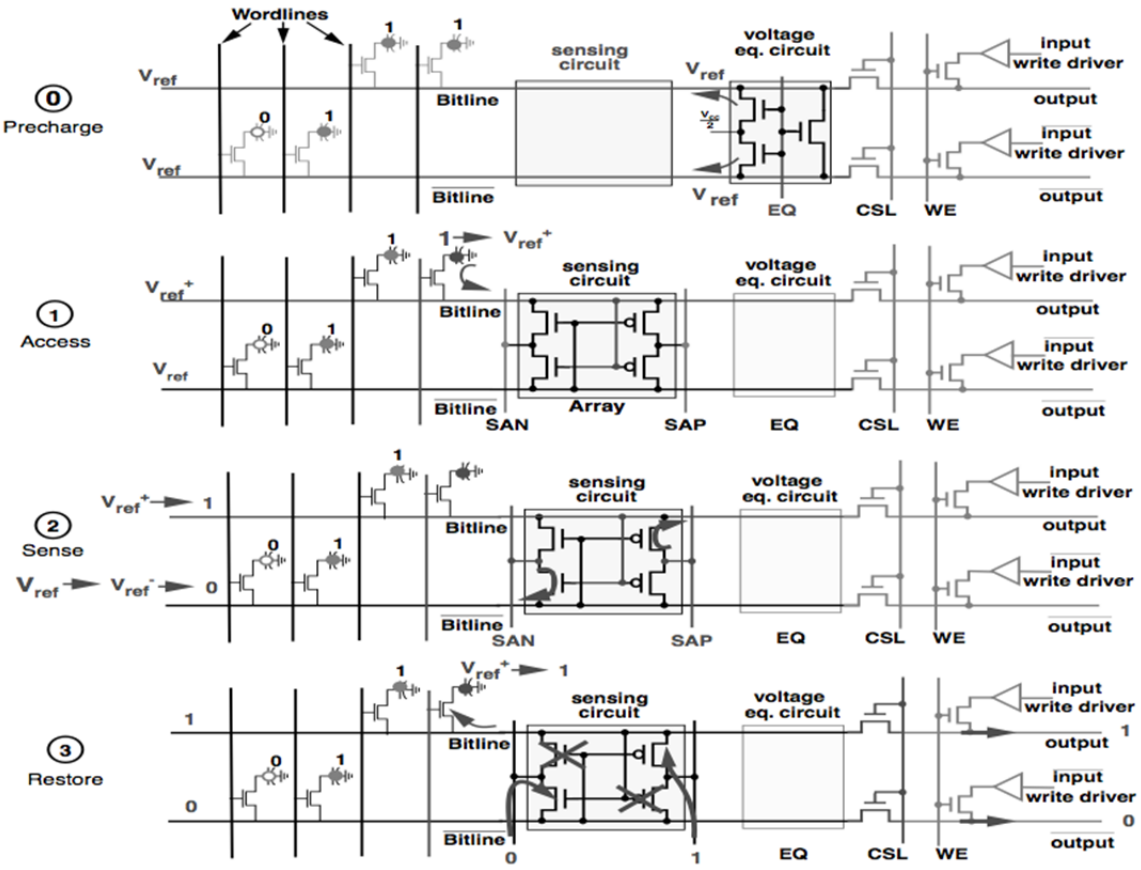


Figura 5.54 – Ilustração do funcionamento dos circuitos de leitura e escrita de numa célula de memória DRAM.

Em resumo, para efectuar a leitura de um dado bit é necessário:

1) Carregar as linhas de bit com $V_{DD}/2$ (com o amplificador de leitura desligado).
2) Desligar o circuito de pré-carga. Como as linhas de bit são longas elas vão manter a carga.
3) Activar a linha de palavra correspondente ao bit que se pretende ler. Isto liga o condensador de célula à linha de bit provocando uma redistribuição de carga entre o condensador e a linha. Como as linhas de bit são longas e os condensadores são pequenos a perturbação da tensão na linha de bit vai ser mínima.
4) Ligar o amplificador de leitura. Isso vai fazer com que uma das linhas de bit da coluna atinja VDD e a outra 0 consoante o valor do bit armazenado.
5) Ler o valor das linhas de bit seleccionadas a partir do endereço de coluna. Podem nessa altura ser lidas vários bits da mesma linha de palavra.
6) Durante a leitura das linhas de bit existe corrente que flui entre o condensador e a linha de bit e que o vai carregar ou descarregar retornando a sua carga ao valor inicial. Devido ao valor elevado de capacidade das linhas de bit este processo é demorado.
7) A linha de palavra é desliga terminando-se o processo de leitura.

No caso da escrita é imposta uma tensão nas linhas de bit que é amplificada pelo amplificador de leitura e que leva ao carregamento ou descarregamento do condensador da linha de palavra seleccionada.

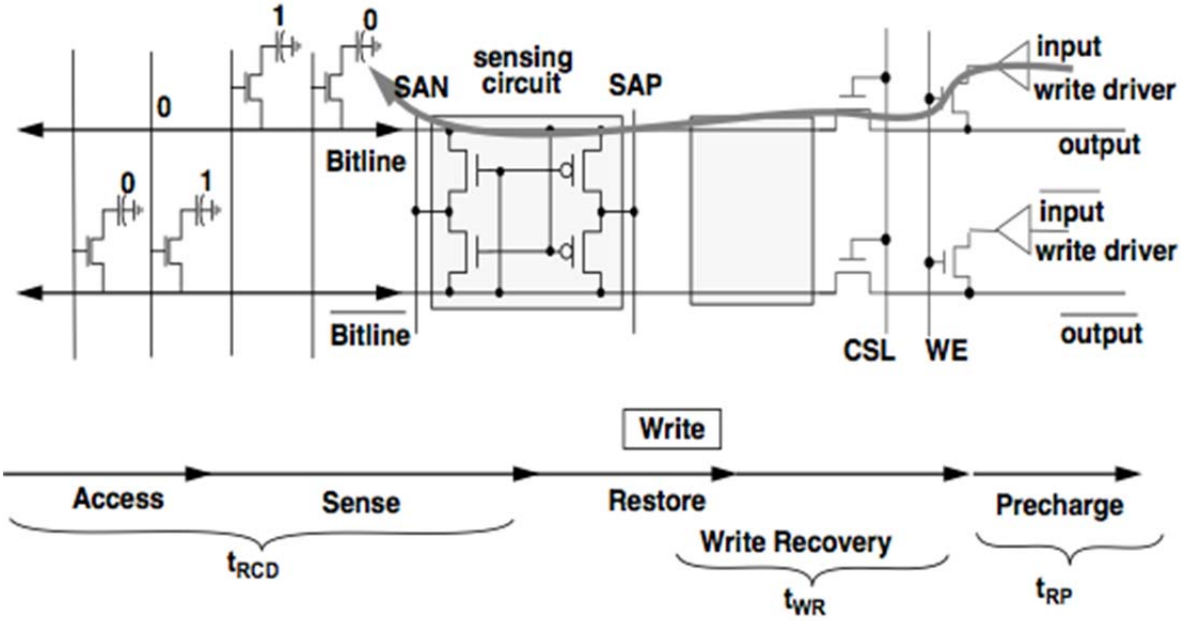


Figura 5.55 – Ilustração da escrita de um bit numa célula de memória DRAM.

Como se viu anteriormente, os bits de um dispositivo de memória estão organizados numa matriz com linhas e colunas. Cada vez que se quer ler ou escrever há que pré-carregar e equalizar todas as linhas de bit. Em memórias muito grandes isso leva a um consumo desnecessário pois muitas dessas linhas provavelmente não serão utilizadas num dado acesso. Por essa razão é comum dividir o dispositivo de memória em várias matrizes independentes chamadas de *bancos*. O endereçamento da informação dentro do dispositivo de memória passa então pela indicação do banco, linha e coluna (Figura 5.56).

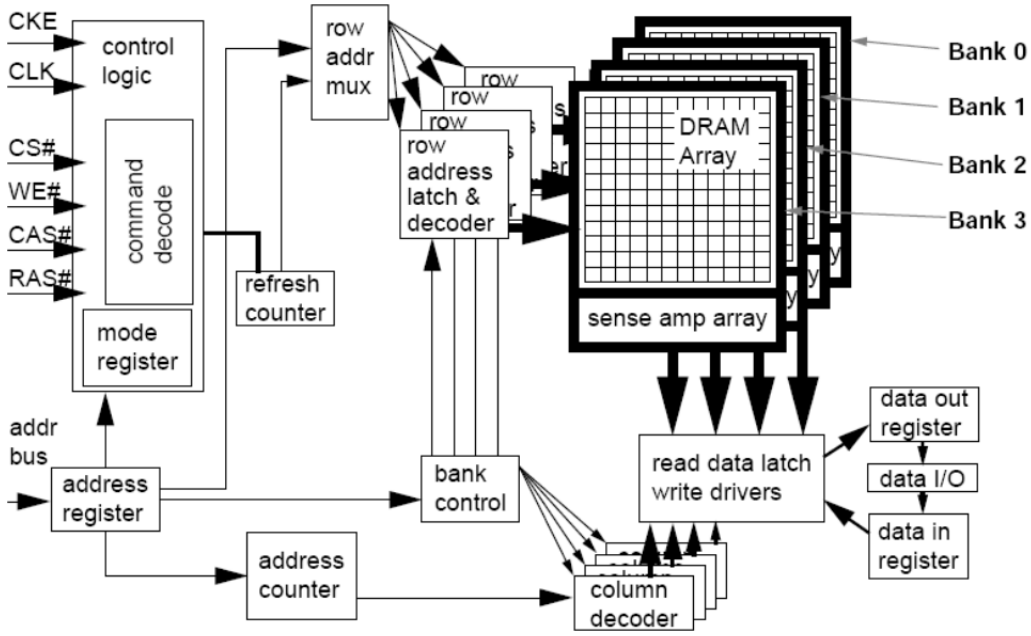


Figura 5.56 – Organização interna de um dispositivo de memória DRAM.

5.4.3 Sinais de Controlo das Memórias DRAM

Ao longo do tempo as memórias DRAM sofreram diversas evoluções com o objectivo de acompanhar o aumento de rapidez dos processadores.

A partir de meados da década de 70 as memórias DRAM eram assíncronas sendo a sua operação controlada pelos sinais RAS e CAS (activos no nível baixo). Quando o nível do sinal RAS transitava do nível alto para o nível baixo a memória lia das linhas de endereço do barramento o endereço da linha que se pretendia aceder. Passado um certo intervalo de tempo o sinal CAS podia transitar do nível alto para o nível baixo indicando que o endereço da coluna que se pretendia aceder já se encontrava no barramento para o dispositivo de memória o ler. Passado um certo intervalo de tempo os dados lidos ficam disponíveis na saída do dispositivo de memória. Para ler outra palavra era necessário repetir este procedimento mesmo que a palavra estivesse no mesmo endereço (Figura 5.57).

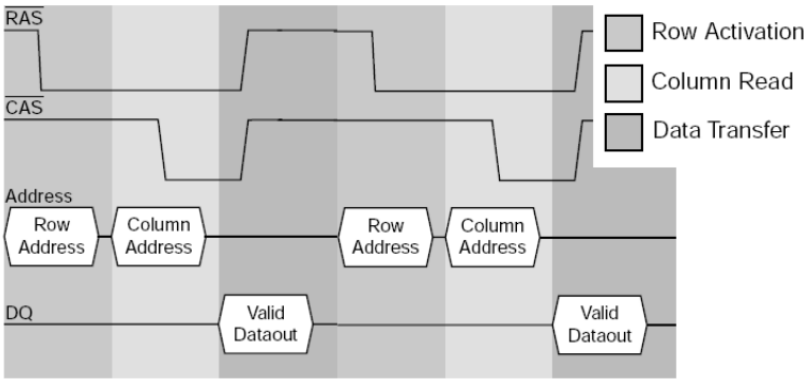


Figura 5.57 – Ilustração dos sinais no barramento de uma memória DRAM assíncrona.

A evolução seguinte deste tipo de memória consiste no modo rápido de página (FPM – ***fast page mode***) que permite a leitura seguida de várias palavras que estejam armazenadas na mesma linha da matriz de memória sem ter que se desactivar e voltar a activar a linha RAS (Figura 5.58).

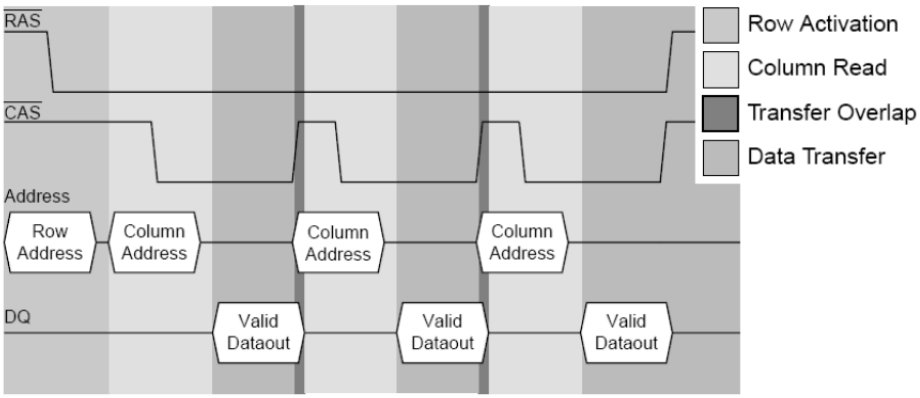


Figura 5.58 – Ilustração dos sinais no barramento de uma memória DRAM assíncrona em modo rápido de página.

Isto é vantajoso porque cada vez que uma linha é activa todas as células de memória são ligadas às suas linhas de bit e a variação da tensão nessas linhas é amplificada pelos amplificadores de leitura de

modo a ser interpretada como sendo um 0 ou um 1. Se a leitura seguinte for de uma palavra que se encontra na mesma linha é vantajoso não ter que desligar as células das linhas, pré-carregar as linhas, voltar a ligar as células às linhas e amplificar a tensão destas. O tempo de acesso é bastante reduzido.

O espaço de memória é organizado de modo a que endereços consecutivos estejam armazenados na mesma linha. Por exemplo, numa memória com 10 palavras por linha e 5 linhas (50 palavras ao todo), as palavras com endereço 0 a 9 estão armazenadas na linha 0, as palavras com endereço 10 a 19 estão armazenadas na linha 1 e assim sucessivamente. Às palavras numa mesma linha dá-se o nome de página. Neste exemplo, portanto, existem 5 páginas (linhas) com 10 palavras cada.

Note-se que neste caso as palavras de uma mesma página (linha) não têm que ser acedidas de forma consecutiva para este modo trazer benefícios pois o endereço da coluna de cada palavra é fornecido com cada acesso (quando a linha CAS está no nível baixo).

Se as palavras desejadas estiverem em linhas diferentes então a existência deste modo de funcionamento (FPM) não trás vantagens.

Uma outra forma de aumentar a rapidez de acesso da memória consiste em usar registos de saída que armazenam os valores de saída dos amplificadores de leitura em vez de estes estarem directamente ligados ao barramento de dados. Neste modo, conhecido por *Extended Data-Out* (EDO), as células de memória podem ser mais rapidamente desligadas das linhas de bit e estas podem mais cedo começar a serem recarregadas para o acesso seguinte pois o registo de saída mantém os dados no barramento para serem lidos pelo processador (Figura 5.59).

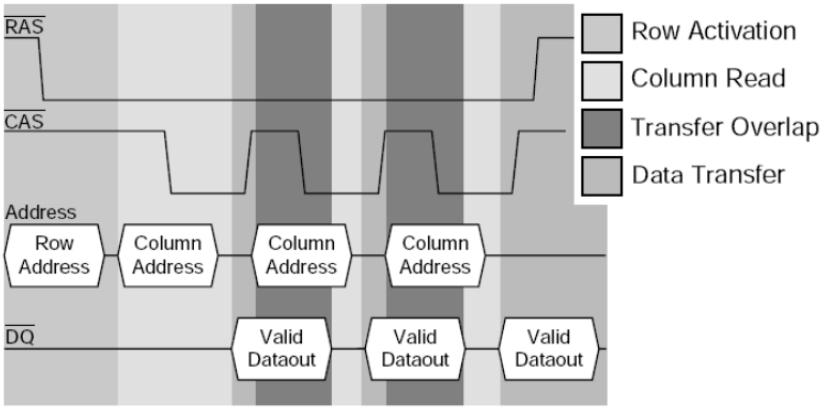


Figura 5.59 – Ilustração dos sinais no barramento de uma memória DRAM assíncrona do tipo EDO.

Uma evolução seguinte deste tipo de memória foi a memória Burst EDO (BEDO). Neste tipo de memória não era necessário fornecer o endereço de cada coluna. Era fornecido o endereço da coluna da primeira palavra desejada e internamente esse endereço era incrementado para que em cada transição do nível alto para o nível baixo do sinal CAS fosse lida uma nova palavra da mesma linha (Figura 5.60).

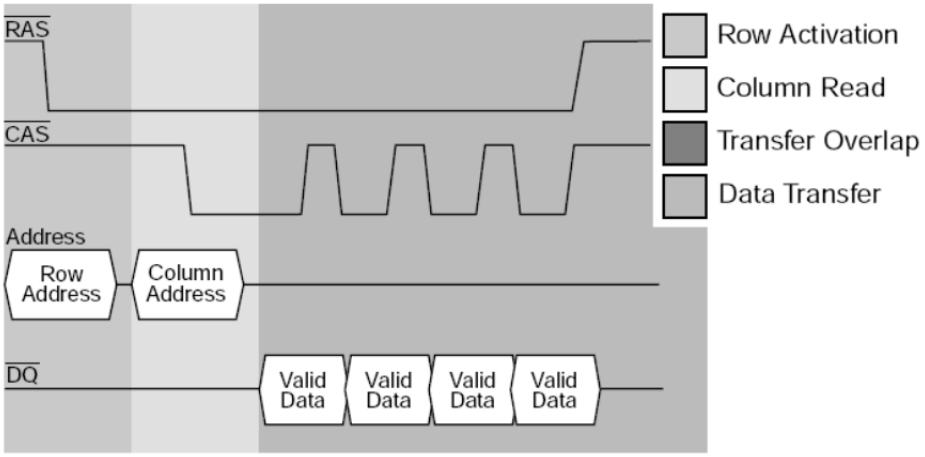


Figura 5.60 – Ilustração dos sinais no barramento de uma memória DRAM assíncrona do tipo BEDO.

Os tipos de memória DRAM apresentados até agora são todos assíncronos na medida que os sinais de controlo RAS e CAS produzem um efeito imediato na memória. A evolução seguinte, já no início da década de 90, consistiu em usar um sinal de relógio para controlar o funcionamento da interface de memória (SDRAM – *synchronous* DRAM). Continuaram a ser usados os sinais de controlo RAS e CAS no entanto a memória só reage ao seu valor nos flancos ascendentes do sinal de relógio (Figura 5.61).

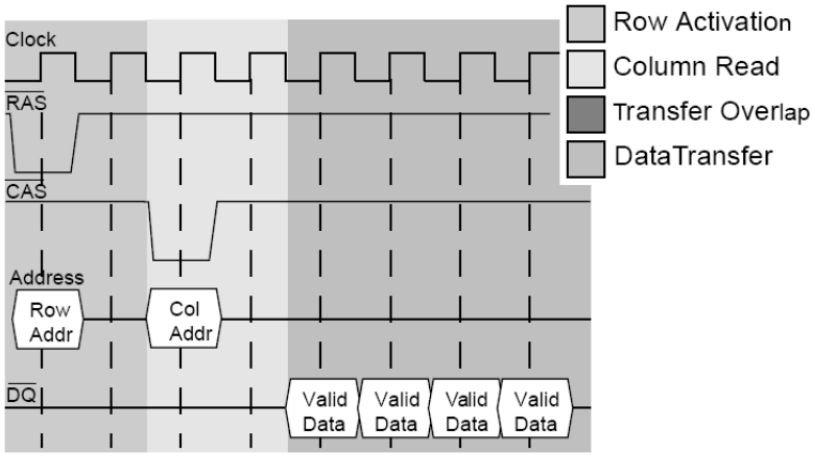


Figura 5.61 – Ilustração dos sinais no barramento de uma memória DRAM síncrona – SDRAM.

Tal como no caso da memória BEDO, existe um contador interno que incrementa o valor do endereço de coluna de modo a serem lidas posições consecutivas de memória. As palavras lidas são disponibilizadas para o exterior também nos flancos ascendentes do sinal de relógio.

As memórias DDR SDRAM (*double data rate* SDRAM) funcionam da mesma forma que as memórias SDRAM mas os dados são disponibilizados nos dois flancos do relógio duplicando a largura de banda (Figura 5.62).

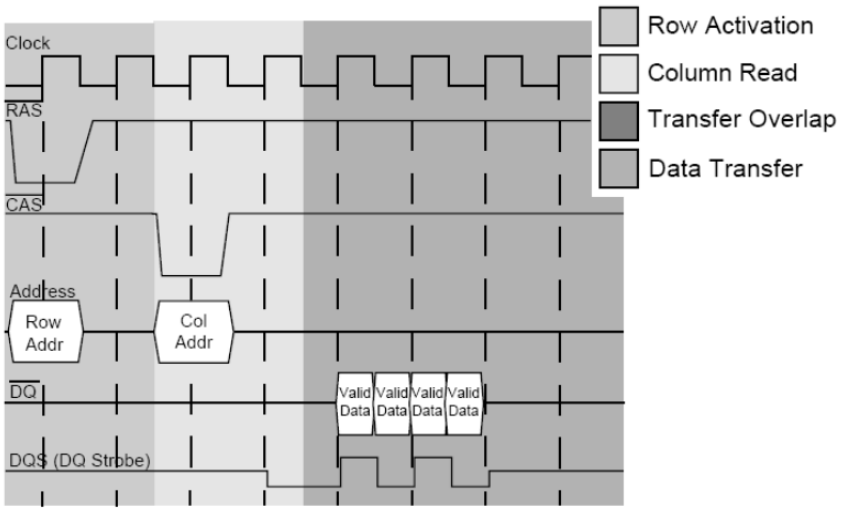


Figura 5.62 – Ilustração dos sinais no barramento de uma memória DRAM síncrona – DDR SDRAM.

5.4.4 Temporizações

Para o correcto funcionamento dos dispositivos de memória há que garantir que entre os flancos dos diversos sinais enviados decorra um intervalo de tempo superior ao valor especificado pelo fabricante. No caso das memórias funcionando de forma assíncrona (DRAM, FPM, EDO, BEDO) esses tempos mínimos são especificados em segundos enquanto no caso das memórias síncronas (SDRAM, DDR SDRAM) esses tempos são especificados em múltiplos do período do sinal de relógio.

Em geral as especificações fornecidas são as seguintes:

- t_{CL} (*CAS Latency*) – Intervalo de tempo entre o envio do sinal CAS e a disponibilização dos respectivos dados.
- t_{RCD} (*RAS to CAS Delay*) – Intervalo de tempo mínimo que tem que haver entre a activação da linha e da coluna de uma determinada posição de memória.
- t_{RP} (*RAS Precharge*) – Intervalo mínimo necessário entre a desactivação do acesso a uma linha e activação do acesso a outra.
- t_{RAS} (*Active to Precharge Delay*) – Intervalo necessário entre um comando de activar linha e a próxima acção do mesmo tipo.
- *CR* (*Command Rate*) – Intervalo que há entre a activação do sinal de selecção do dispositivo (CS – *chip select*) e qualquer outro comando.

Num computador pessoal os tempos usados entre os diversos sinais podem ser escolhidos pelo utilizador na BIOS. Os valores apresentados por defeitos são aqueles que o computador obtém dos próprios módulos de memória onde existe uma memória não volátil (EEPROM) onde o fabricante programa esses valores (Figura 5.63).

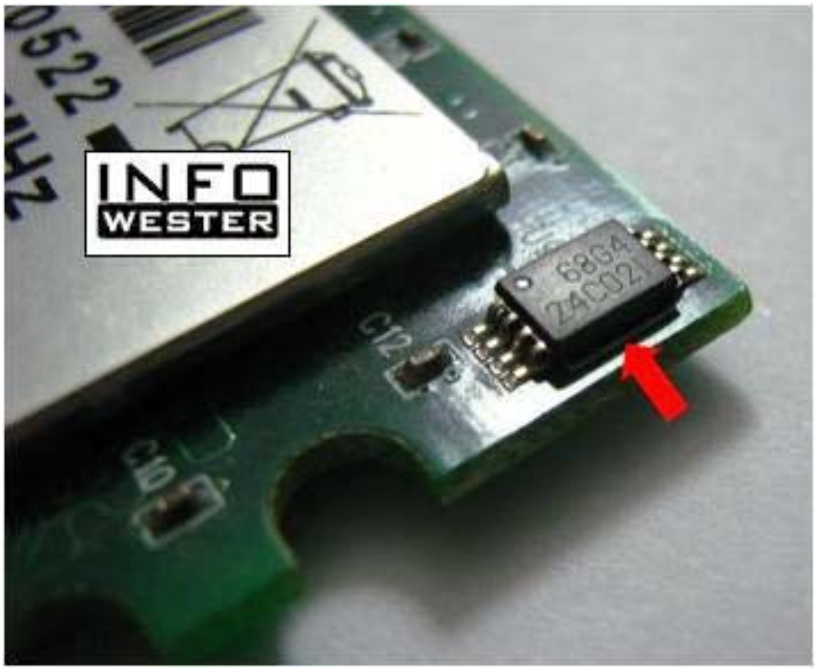


Figura 5.63 – Fotografia da EEPROM usada pelo fabricante para armazenar diversa informação num módulo de memória.

A Figura 5.64 mostra um exemplo das temporizações de memória usadas num computador pessoal. Note-se como o tempo de ciclo (t_{RAS}) é a soma do tempo de pré-carga das linhas de bit (t_{RP}), o atraso entre a linha RAS e a linha CAS (t_{RCD}) e o tempo entre a activação da linha CAS e a disponibilização dos dados (t_{CL}).

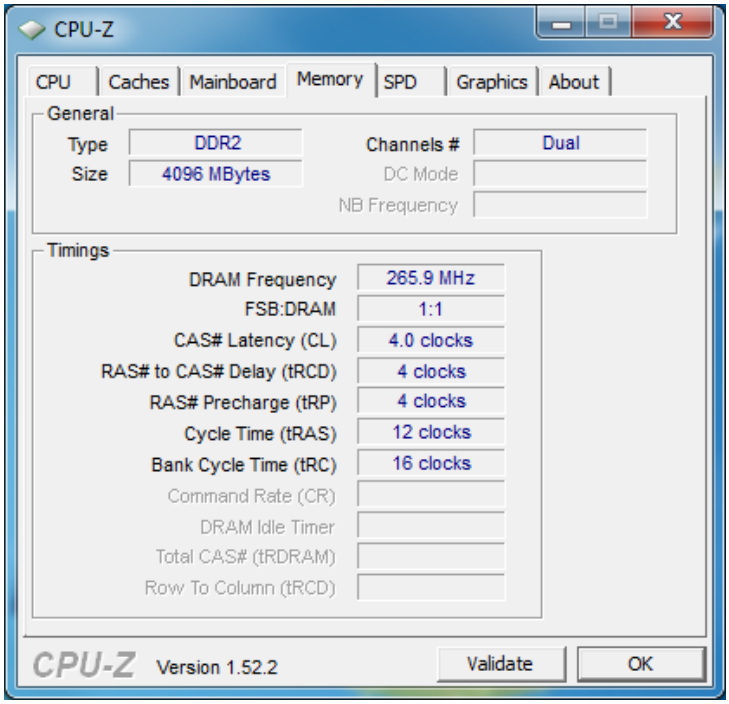
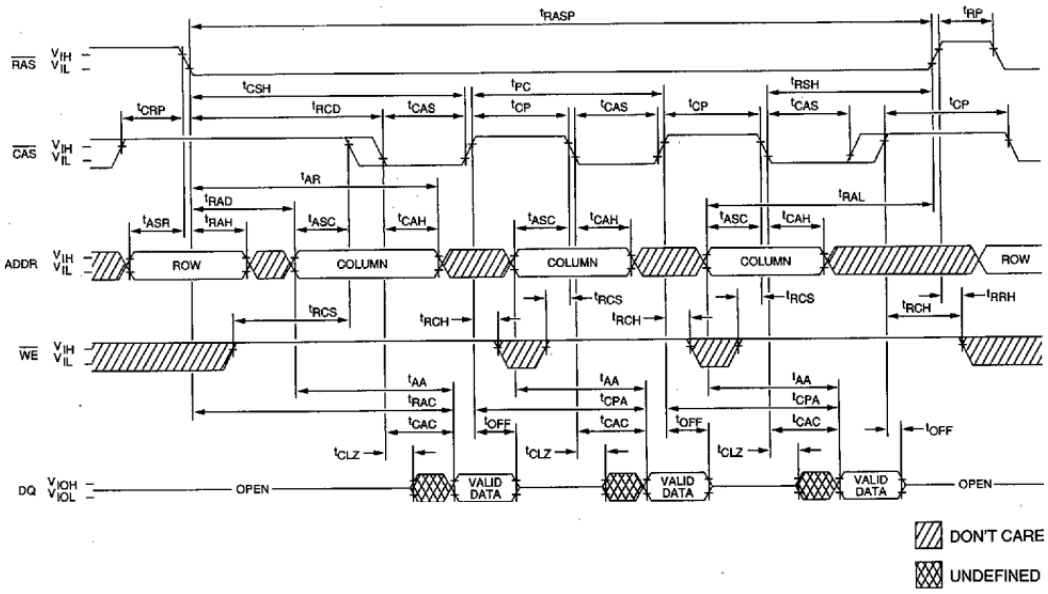


Figura 5.64 – Valores das temporizações de memória de um computador pessoal fornecidas pela aplicação CPU-Z.

Veja-se agora como exemplo qual a máxima largura de banda que se consegue com os módulos de memória DRAM da Micron, modelo MT16D232, através do uso do modo de página rápido para

transferência de blocos de 32 bytes. Nesse modo vários bytes da mesma linha podem ser transferidos sem ter que se fornecer individualmente o endereço de linha (Figura 5.65).

FAST-PAGE-MODE READ CYCLE



FAST PAGE MODE TIMING PARAMETERS

	-6		-7		
SYM	MIN	MAX	MIN	MAX	UNITS
tAA		30		35	ns
tAR	45		50		ns
tASC	0		0		ns
tASR	0		0		ns
tCAC		15		20	ns
tCAH	10		15		ns
tCAS	15	10,000	20	10,000	ns
tCLZ	0		0		ns
tCP	10		10		ns
tCPA		35		40	ns
tCRP	10		10		ns
tCSH	60		70		ns
tOFF	3	15	3	20	ns

	-6		-7		
SYM	MIN	MAX	MIN	MAX	UNITS
tPC	35		40		ns
tRAC		60		70	ns
tRAD	15	30	15	35	ns
tRAH	10		10		ns
tRAL	30		35		ns
tRASP	60	100,000	70	100,000	ns
tRCD	20	45	20	50	ns
tRCH	0		0		ns
tRCS	0		0		ns
tRP	40		50		ns
tRRH	0		0		ns
tRSH	15		20		ns

Figura 5.65 – Extracto da folha de especificações dos módulos de memória DRAM da Micron, referência MT16D232, mostrando as temporizações em modo rápido de página.

Observando o diagrama temporal fornecido na folha de especificações da memória para o caso de uma leitura usando o modo rápido de página, conclui-se que a leitura de, por exemplo, um bloco de 32 bytes requer:

1) a colocação do endereço de linha no barramento de endereços;

2) a activação do sinal RAS;
3) a colocação do endereço de coluna no barramento de endereços;
4) a activação do sinal CAS;
5) a leitura dos dados presentes no barramento de dados;
6) a desactivação da linha CAS;
7) a repetição dos passos 3 a 5 para a leitura de cada um dos 32 bytes.

Depois de 32º flanco descendente do sinal CAS há que desactivar o sinal RAS de modo a se poder começar a transferir outro bloco repetindo-se as mesmas operações (Figura 5.66).

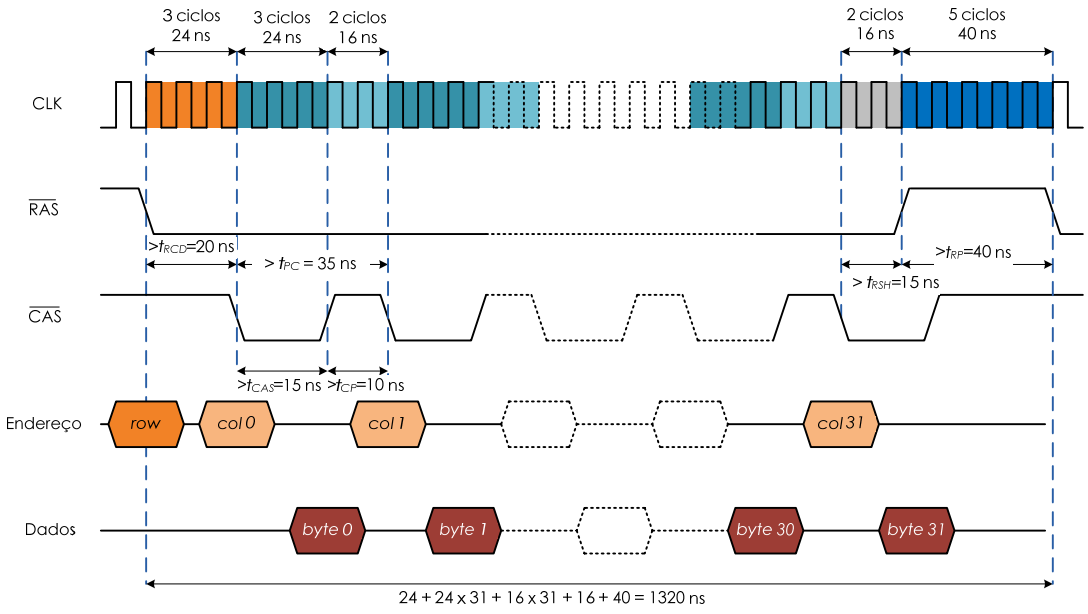


Figura 5.66 – Diagrama temporal dos sinais de controlo da memória DARM do exemplo.

A activação e desactivação dos sinais de controlo, nomeadamente do sinal RAS e do sinal CAS, requerem que se cumpram intervalos de tempo mínimos consoante o indicado na folha de especificações da memória.

Suponha-se que o sistema de memória a construir utiliza um controlador síncrono com o processador. Isto significa que o controlador executa uma máquina de estados finita havendo lugar a uma transição de estado a cada ciclo de relógio. Considerando, por exemplo, uma frequência de 125 MHz, as transições de estado acontecem, no mínimo, intervaladas de 8 ns (1/125 μs). Os sinais de saída do controlador ligados às SIMMs são portanto alterados de cada vez que se entra num novo estado. Isso implica que o intervalo mínimo entre dois flancos do mesmo ou de sinais diferentes é de 8 ns. Tendo em conta esta resolução temporal e os tempos mínimos impostos pela memória, há que determinar o número de ciclos de relógio (transições de estados da máquina de estado finita do controlador) a utilizar entre os diferentes flancos dos sinais de controlo (RAS e CAS).

Considera-se que o início da transferência acontece quando o sinal RAS é activado (flanco descendente). Entre esse flanco e o flanco descendente do sinal CAS tem que existir um intervalo de

tempo igual ou superior a 20 ns (t_{RCD} – *RAS to CAS delay*). Isso corresponde portanto a 3 ciclos de relógio pois 3 x 8 ≥ 20.

De seguida há que activar o sinal CAS que deve manter-se activo durante 15 ns (t_{CAS} – *CAS pulse width*) o que corresponde a 2 ciclos de relógio. De seguida o sinal CAS é desactivado e tem que se manter desactivado durante 10 ns (t_{CP} – *CAS precharge time*) o que corresponde também a 2 ciclos de relógio. Esses dois tempos levariam a crer que o ciclo de activação e desactivação do sinal CAS (necessário para transferir cada byte) poderia ocupar unicamente 4 ciclos de relógio. Há, no entanto, que ter em atenção que a folha de especificações indica que deve ocorrer um intervalo mínimo de 35 ns entre activações consecutivas do sinal CAS (t_{PC} – *FAST-PAGE-MODE READ or WRITE cycle time*). Isso corresponde a 5 ciclos de relógio. Escolheu-se portanto usar 3 ciclos de relógio para o intervalo de tempo que o sinal CAS está desactivado e 2 ciclos para o intervalo de tempo em que está activado.

Depois do 32º flanco descendente do sinal CAS, quando é realizada a leitura do 32º byte, há que desactivar o sinal RAS. O tempo que deve passar entre uma coisa e outra é de 15 ns (t_{RSH} – *RAS hold time*). Isso corresponde a 2 ciclos de relógio.

Finalmente, para voltar a activar de novo o sinal RAS para dar início à transferência de mais um bloco de dados, há que esperar 40 ns (t_{RP} – *RAS precharge time*). Isso traduz-se em mais 5 ciclos de relógio.

O número de ciclos de relógio a usar para cada fase da leitura dos dados está resumido no diagrama temporal apresentado (3-3-2-2-5). Note-se que o 2º e o 3º número nesta sequência são repetidos 31 vezes para transferência do byte 0 ao byte 30. O tempo total necessário para a leitura de um bloco de 32 bytes, em ciclos de relógio, é portanto

$$N = 3 + (3+2) \times 31 + 2 + 5 = 165 \text{ ciclos}. \quad (24)$$

Em segundos isso equivale a 1320 ns (165 x 8). Conseguem-se assim transferir 32 bytes a cada 1320 ns. Isso equivale à transferência de $24{,}24 \times 10^6$ bytes por segundo. A largura de banda é portanto 23,12 MB/s ($24{,}24 \times 10^6$ / 1024 / 1024).

5.4.5 Refrescamento

As memórias do tipo DRAM têm a particularidade de que a informação nelas contida é perdida com o tempo mesmo mantendo-se a alimentação. Isso acontece porque os bits são representados através da carga eléctrica armazenada num condensador. Não sendo possível isolar de forma perfeita o condensador do circuito que o rodeia, a carga acaba por fluir para fora dele com o passar do tempo.

Há portanto a necessidade de periodicamente restabelecer o nível de carga de cada condensador da memória. Este processo, a que se dá o nome de *refrescamento*, é realizado normalmente pelo controlador de memória existindo 3 formas de o efectuar:

- Só-RAS
- CAS-antes-de-RAS

- Escondido

O refrescamento Só-RAS (RAS-ONLY) é efectuado activando-se uma linha através do sinal de controlo RAS, tal como se faz num acesso normal à memória, e fornecendo-se o endereço da linha a activar (Figura 5.67). O controlador mantém um contador com o endereço da linha a refrescar e incrementa-o de modo a percorrer todas as linhas da memória. Os acessos à memória podem ter que ser atrasados de modo a terminar o processo de refrescamento.

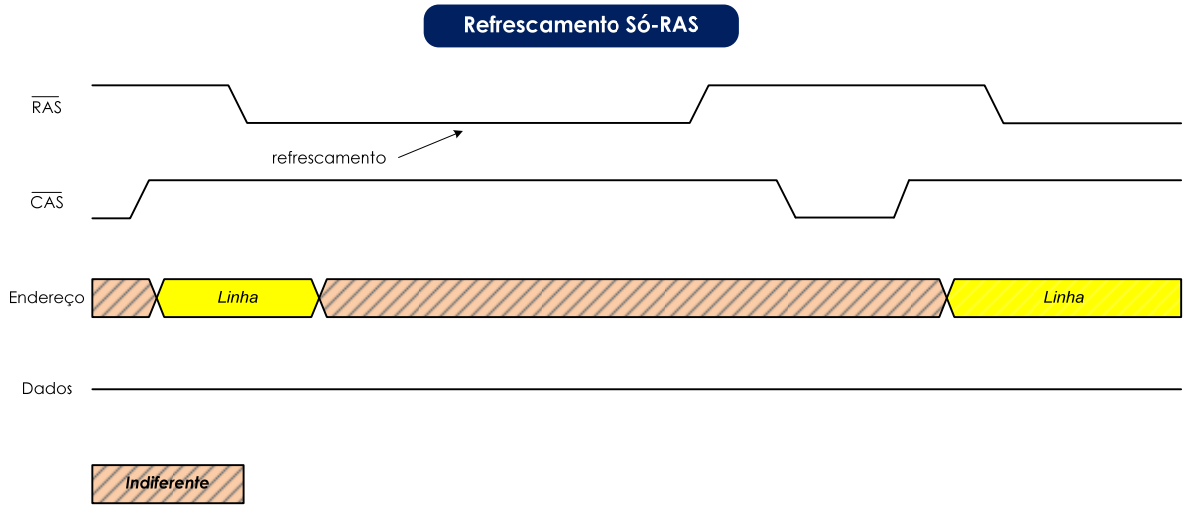


Figura 5.67 – Diagrama temporal dos sinais de controlo durante o refrescamento SÓ-RAS de uma memória DRAM.

Para evitar ter que se usar um contador externo para determinar o endereço na linha a refrescar é possível implementar esse contador dentro do módulo de memória. Neste caso o controlador só tem que indicar ao módulo de memória que deve refrescar uma linha e é deixado ao próprio módulo a geração do endereço da linha a refrescar em cada momento. Essa indicação é feita activando-se o sinal de controlo CAS antes de se activar o sinal RAS, o que não é permitido num acesso normal ao conteúdo da memória (Figura 5.68). Este modo chama-se por isso refrescamento CAS-antes-de-RAS (*CAS-before-RAS*). Este é o modo mais utilizado hoje em dia, em especial nas memórias SDRAM.

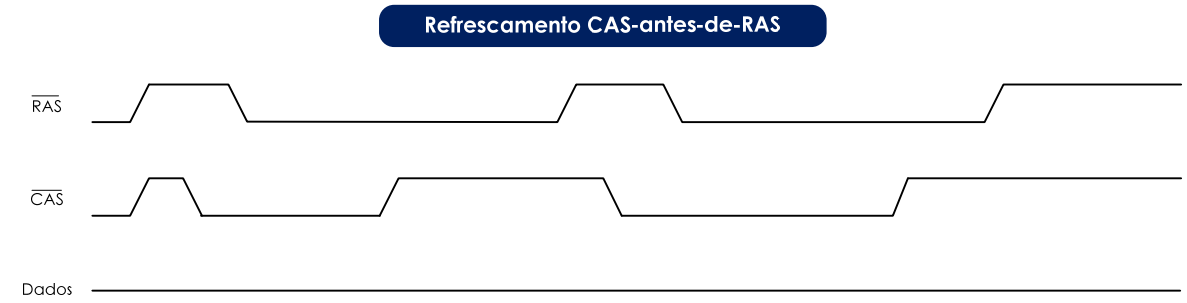


Figura 5.68 – Diagrama temporal dos sinais de controlo durante o refrescamento CAS-antes-de-RAS de uma memória DRAM.

O modo de refrescamento escondido (*hidden refresh*) é uma melhoria do modo CAS-antes-de-RAS de que permite que se efectue um refrescamento a seguir a uma leitura normal da memória de tal forma

que os dados de saída permanecem válidos enquanto outras linhas são refrescadas. Num acesso normal a linha RAS é activada primeiro do que a linha CAS. Depois de disponibilizados os dados na saída a linha CAS é desactivada sendo seguida pela desactivação da linha RAS. Se, no entanto a linha CAS for mantida activa e a linha RAS for desactivada, é possível voltar a activar a linha RAS e no fundo realizar um refrescamento CAS-antes-de-RAS enquanto os dados de saída permanecem válidos (Figura 5.69).

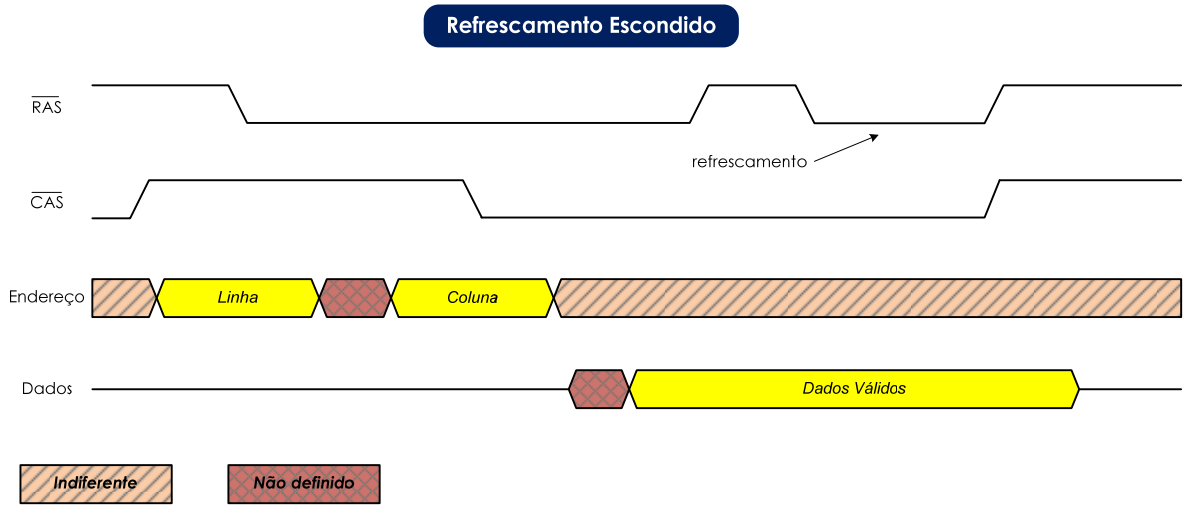
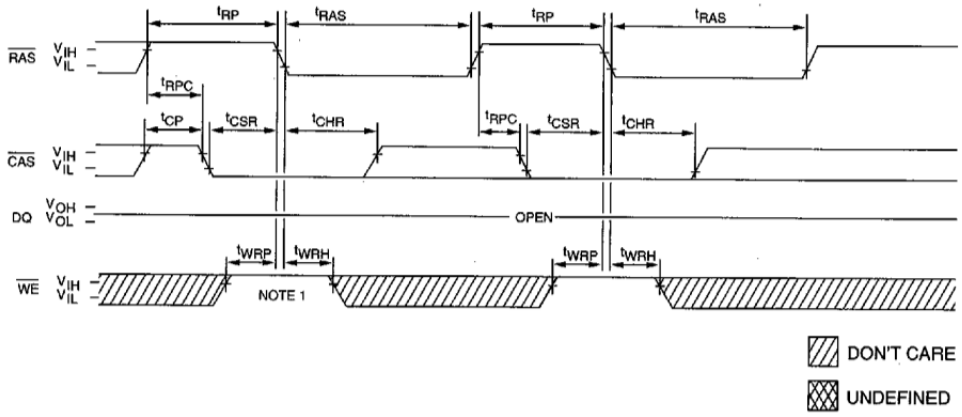


Figura 5.69 – Diagrama temporal dos sinais de controlo durante o refrescamento escondido de uma memória DRAM.

O refrescamento da memória deve ser efectuado periodicamente a um ritmo especificado pelo fabricante e que é em geral de alguns milisegundos.

Os cálculos efectuados na secção anterior relativos à largura de banda máxima que era possível obter para a transferência de blocos de 32 bytes não tiverem em conta o tempo necessário para efectuar o refrescamento da memória. Se considerarmos o refrescamento do tipo CAS-antes-de-RAS observa-se na folha de especificações dos módulos de memória (Figura 5.70) que o refrescamento de uma linha necessita de pelo menos 40 ns durante o qual o sinal RAS está inactivo (t_{RP} – *RAS precharge time*), o que corresponde a 5 ciclos de relógio, e mais 60 ns durante o qual o sinal RAS está activo (t_{RAS} – *RAS pulse width*) o que corresponde a pelo menos 8 ciclos de relógio. Ao todo são preciso pois 13 (5 + 8) ciclos de relógio o que corresponde a 104 ns.

CBR REFRESH CYCLE [29]
(Addresses = DON'T CARE)



NOTE: 1. Although $\overline{WE}$ is a "don't care" at $\overline{RAS}$ time during an access cycle (READ or WRITE), the system designer should implement $\overline{WE}$ HIGH for tWRP and tWRH. This design implementation will facilitate compatibility with future EDO DRAMs.

FAST PAGE MODE AND EDO PAGE MODE TIMING PARAMETERS

	-6		-7		
SYM	MIN	MAX	MIN	MAX	UNITS
tASR	0		0		ns
tCHR	10		10		ns
tCP	10		10		ns
tCRP	10		10		ns
tCSR	10		10		ns
tRAH	10		10		ns
tRAS	60	10,000	70	10,000	ns

	-6		-7		
SYM	MIN	MAX	MIN	MAX	UNITS
tRC	110		130		ns
tRP	40		50		ns
tRPC (FPM)	0		0		ns
tRPC (EDO)	5		—		ns
tWRH	10		10		ns
tWRP	10		10		ns

Figura 5.70 – Extracto da folha de especificações dos módulos de memória DRAM da Micron, referência MT16D232, mostrando as temporizações para refrescamento do tipo CAS-antes-de-RAS.

O refrescamento das 1024 linhas que a memória DRAM tem demora portanto 13312 (13 x 1024) ciclos de relógio (106,496 μs). Segundo o fabricante esse refrescamento tem de ser executado pelo menos uma vez em cada 16 ms. Há então que planear como esse tempo é usado para transferir dados (blocos de 32 bytes) e para efectuar o refrescamento.

Em 16 ms existem 2 milhões de ciclos de relógio. Retirando os 13312 que são necessários para o refrescamento sobram 1986688 ciclos. Tendo em conta que a transferência de 32 bytes requer 165 ciclos de relógio, conseguem-se efectuar 12040 transferências de bloco (1986600 ciclos de relógio). Ficam a sobrar 2000000 – 13312 – 1986600 = 88 ciclos de relógio. A transferência dos 1204 blocos de dados mais o refrescamento das 1024 linhas demora 1999912 (1986600 + 13312) ciclos de relógio, ou seja, 15,999296 ms. Nesse período de tempo conseguem-se transferir 385280 bytes (Figura 5.71). A largura de banda é pois de aproximadamente 24081059 bytes/s, ou seja, 22,97 MB/s.

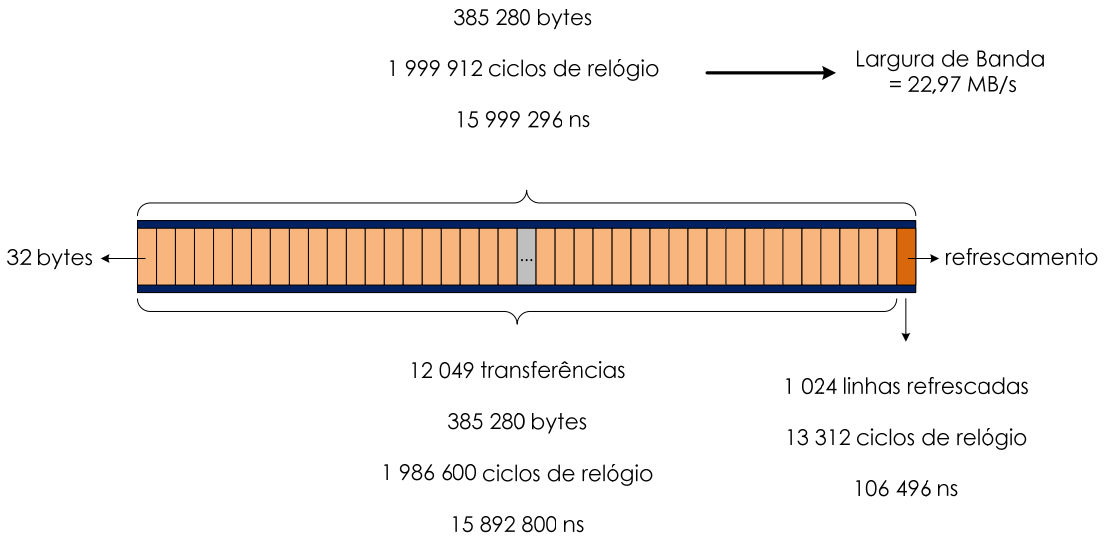


Figura 5.71 – Esquema mostrando as transferências efectuadas e a sua duração no exemplo dado.

5.5 Memória Tampão

5.5.1 Organização da Memória

Como foi visto nas secções anteriores, as memórias mais rápidas que existem hoje em dia (SRAM e DRAM) são baseadas na tecnologia CMOS e fabricadas em circuitos integrados. Estes dois tipos de memórias são voláteis e de acesso aleatório.

O papel da memória num computador moderno é o de armazenar os dados e as instruções dos programas que se pretende sejam executados pelo processador. Esta divisão entre processamento e armazenamento de informação trás o problema da transferência dessa mesma informação entre o processador e a memória. A rapidez do processador, o tempo de acesso à memória e o ritmo de transmissão da ligação entre os dois condicionam directamente o desempenho do computador. Hoje em dia o processador é, em geral, o componente mais rápido tendo normalmente que esperar que a informação vá e venha da memória.

A escolha adequada, em termos do projecto de um computador, seria o de uma memória com o menor tempo de acesso possível e a sua colocação o mais perto do processador possível de modo a minimizar o comprimento das linhas de barramento. Como se verá no próximo capítulo, dedicado aos barramentos, no caso da interface entre processador e memória escolhe-se, em geral, ter um barramento paralelo, que permite transmitir vários bits ao mesmo tempo de forma a aumentar o ritmo de transmissão (número de bits por segundo).

A escolha ideal de memória, tendo em conta a rapidez de funcionamento do computador é a do tipo SRAM que tem tempos de acesso da ordem dos nanossegundos. A rapidez, no entanto, não é a única métrica a ter em conta. Muitas vezes a quantidade de informação que é possível armazenar na memória é mais importante. Nessa perspectiva a memória DRAM trás vantagens devido ao menor

tamanho da células de memória e consequente reduzido preço de fabricação quando comparada com uma memória SRAM do mesmo tamanho. Nos computadores pessoais, por exemplo, é comum ter-se hoje em dia vários gigabytes de memória RAM que, se fossem do tipo SRAM, teriam um custo muito elevado.

A solução adoptada no caso dos computadores pessoais é o uso combinado de memórias SRAM e DRAM. Esta última é usada como memória principal, de grande capacidade, enquanto uma memória do tipo SRAM é usada entre o processador e a memória principal funcionando como um tampão (*cache*) como ilustra a Figura 5.72. Muitas vezes essa memória tampão é integrada dentro do próprio circuito integrado do processador de forma a maximizar o ritmo de transferência com o processador em si.

Figura 5.72 – Ilustração da utilização de uma memória tampão entre o processador e a memória principal.

O tamanho da memória tampão é em geral várias ordens de grandeza menor do que o tamanho da memória principal. A primeira tem em geral alguns megabytes de capacidade enquanto a segunda alguns gigabytes. Qual é então a vantagem de se ter uma memória rápida entre os processador e a memória principal se esta só é capaz de guardar 0,1% da informação existente na memória principal? Seria de prever que a maior parte das vezes que o processador necessitar de buscar informação à memória essa estará armazenada na memória principal. Com certeza que não seria economicamente viável utilizar uma memória que já de si é cara para só ser usada 0,1% do tempo. Na prática, no entanto, a existência da memória tampão, mesmo com esse tamanho reduzido, é bastante vantajosa. Isso acontece porque a informação que o processador normalmente necessita de aceder num determinado instante não está numa posição completamente aleatória de memória mas está, com uma probabilidade significativa, em certas localizações de memória específicas que podem ser previstas. Isso acontece devido à forma como os programas são criados. Na maior parte das vezes os programas são elaborados tendo em conta um conjunto de passos que se sucedem um ao outro. Isso faz com que, depois de acedido um certo item da memória, seja provável que esse item seja necessário novamente assim como os itens adjacentes. Este facto pode ser enunciado num princípio conhecido como *Princípio da Localidade*:

Princípio da Localidade

Num dado instante, um programa acede a uma pequena parcela do espaço de endereçamento.

Há dois tipos de localidade:

- Localidade temporal: Se um item é referenciado, existe forte probabilidade que seja referenciado num futuro próximo (e.g., ciclos de programa, reutilização de dados).
- Localidade espacial: Se o item num determinado endereço é referenciado, existe forte probabilidade que sejam referenciados items em endereços próximos e num futuro próximo (e.g., código sequencial, acessos a *arrays*).

A organização de memória ilustrada na Figura 5.72 é um caso simples onde é usada uma memória tampão. Há casos em que não é usada nenhuma memória tampão ou em que a memória principal é do tipo SRAM, como no caso dos microcontroladores. Existem também situações em que existem mais do que um nível de memória tampão, normalmente designadas por L1 e L2 (*level* 1/2) e implementadas dentro e fora do processador respectivamente. A Figura 5.73 ilustra uma organização genérica da memória num computador onde são utilizados dois níveis de memória tampão. Está também representada uma memória secundária que é normalmente não volátil, onde os programas e dados são guardados de forma permanente. Esse tipo de memória é em geral um disco rígido, no caso dos computadores pessoais ou uma memória EEPROM ou Flash no caso de sistemas embebidos.

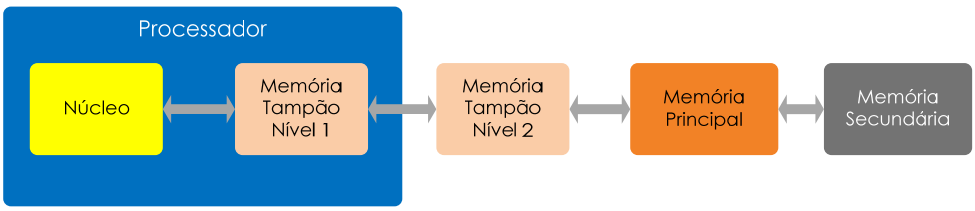


Figura 5.73 – Ilustração genérica da forma como está organizada a memória num computador.

A organização da memória pode ter um qualquer número de níveis. As métricas correspondentes aos diferentes níveis são as ilustradas na Figura 5.74. Uma organização deste tipo é fruto das limitações das diferentes tecnologias existentes para a construção de memórias. Se fosse possível integrar dentro de um circuito integrado memórias com tempos de acesso inferiores ao nanosegundo, capacidades de vários terabytes a um custo de alguns Euros por Terabyte então não seria necessário haver diferentes níveis de memória no que diz respeito ao desempenho do computador. Isso não significa que não fossem usados diferentes níveis por outras razões como a comodidade ou flexibilidade de utilização.

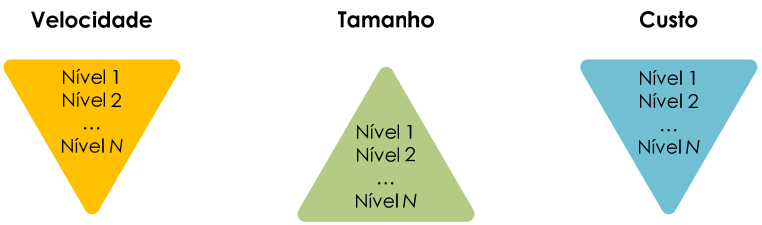


Figura 5.74 – Comparação dos diferentes níveis de memória em termos das métricas velocidade, tamanho e custo.

A Figura 5.75 mostra a arquitectura típica de um microcontrolador onde se pode observar os diferentes tipos de memórias usados.

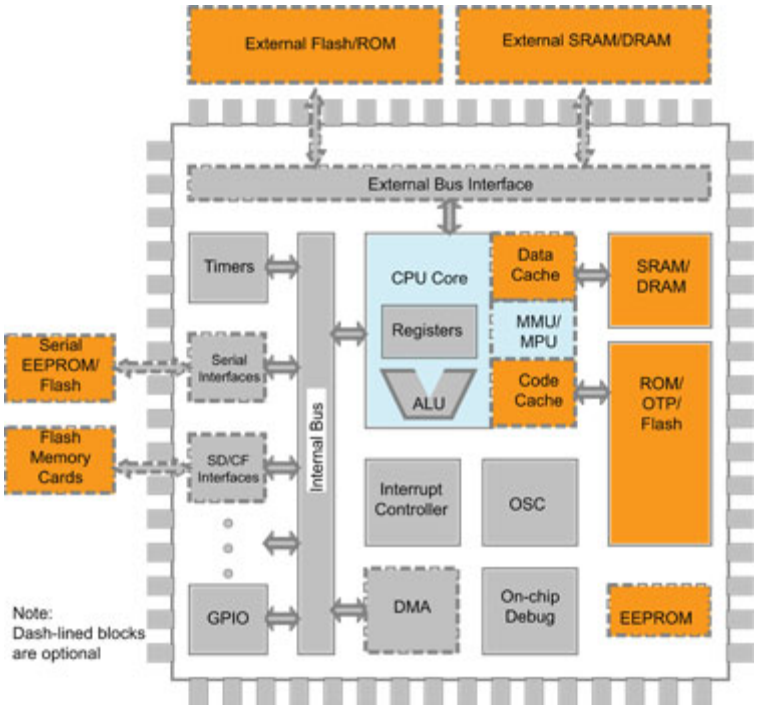


Figura 5.75 – Arquitectura de memória de um microcontrolador [2].

5.5.2 Organização da Memória Tampão

O Princípio da Localidade determina qual a informação a armazenar na memória tampão. A estratégia mais simples seria a de, cada vez que o processador tenta aceder a uma certa posição de memória, copiar da memória principal para a memória tampão um bloco de dados consecutivo contendo o dado desejado. A Figura 5.76 ilustra esta estratégia para um caso em que a memória principal tem 16 bytes e a memória tampão tem 4 bytes (mais 2 bits para a etiqueta). Isto faz com que só um quarto da memória principal possa estar na memória tampão de cada vez.

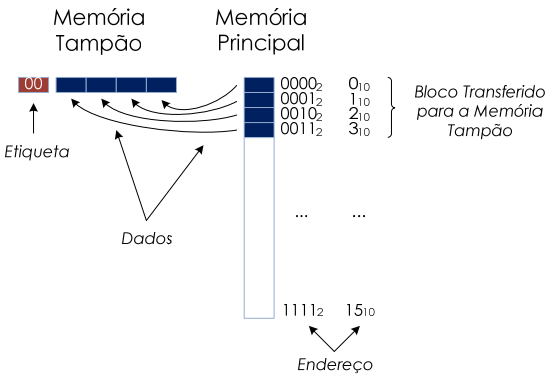


Figura 5.76 – Ilustração do preenchimento da memória tampão com um conjunto consecutivos de palavras da memória principal (endereços 0 a 3).

No caso da Figura 5.76 o byte acedido pelo processador foi, por exemplo, o byte no endereço 2. Se o conteúdo da memória no endereço acedido for de novo pedido pelo processador então poderá ser fornecido pela memória tampão não havendo necessidade que a memória principal forneça esse conteúdo (localidade temporal). Por outro lado, se de seguida forem pedidos os conteúdos nas posições próximas da memória eles se encontrem também na memória tampão já que foi transferido um conjunto de dados consecutivos da memória principal (bloco) para a memória tampão (localidade espacial).

Se de seguida for pedido pelo processador o dado contido no endereço 9 de memória, ele não vai ser encontrado na memória tampão – falha na memória tampão (*cache miss*). Ele terá que ser acedido da memória principal. Ao mesmo tempo os dados guardados na memória tampão são descartados e um novo bloco de dados, desde o endereço 8 ao11 é armazenado na memória tampão.

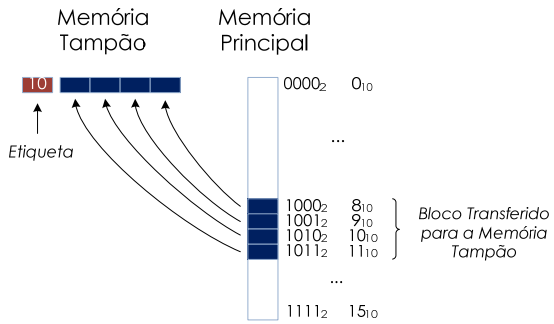


Figura 5.77 – Ilustração do preenchimento da memória tampão com um conjunto consecutivos de palavras da memória principal (endereços 8 a 11).

Para identificar qual o bloco de dados da memória principal está guardado na memória tampão num dado instante é usada uma etiqueta também guardada na memória tampão. Essa etiqueta identifica o endereço do primeiro byte desse bloco na memória principal. No caso do exemplo dado na Figura 5.76 e na Figura 5.77, existem 4 blocos diferentes sendo portanto necessários 2 bits para identificar esse bloco – a etiqueta tem 2 bits de comprimento. Esses dois bits são os 2 bits mais significativos do endereço das palavras desse bloco na memória. Neste exemplo os 2 bits menos significativos do

endereço são usados para identificar qual dos bytes guardado na memória tampão se pretende aceder. Por exemplo, se o endereço fornecido para uma leitura de memória fosse o 1001 (9_{10}), os dois bits mais significativos são comparados com os dois bits da etiqueta guardada na memória tampão para determinar se esse dado se encontrava armazenado nessa memória. Os dois bits menos significativos são usados para escolher qual dos 4 bytes armazenados deve ser devolvido ao processador. O circuito electrónico usado para realizar esta função e o da Figura 5.78.

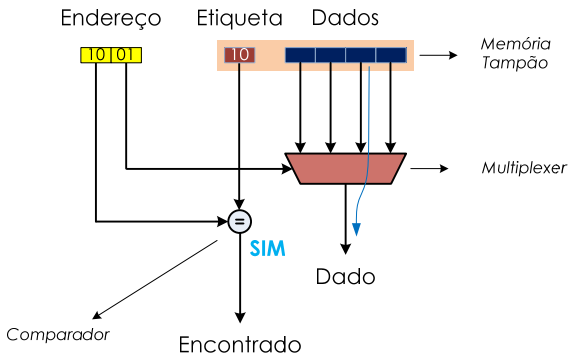


Figura 5.78 – Circuito de acesso à memória tampão.

Se a memória tampão tiver um tamanho considerável (kilobytes ou megabytes) esta estratégia deixa de ser a melhor pois é cada vez menos provável que sejam necessários num futuro próximo os dados guardados na memória perto do fim do bloco. Nessa situação, que é comum na prática, faz mais sentido guardar vários blocos pequenos em vez de um bloco grande como ilustrado na Figura 5.79. Neste exemplo a memória tampão continua a ter um tamanho de 4 bytes (mais as etiquetas) mas agora estão organizados em dois conjuntos de 2 bytes cada. Em cada um desses conjuntos é possível guardar dois bytes da memória principal.

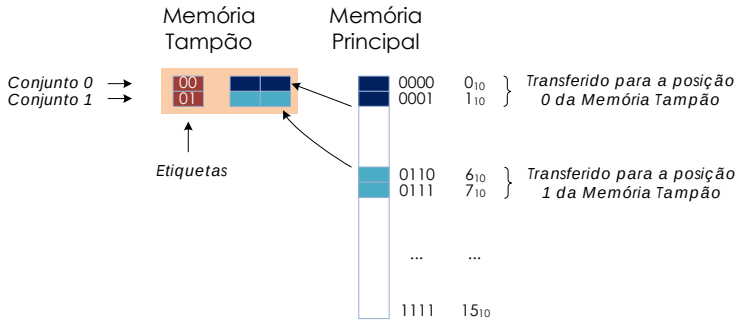


Figura 5.79 – Ilustração do preenchimento da memória tampão constituída por dois conjuntos.

Neste caso o bit menos significativo do endereço é usado para identificar qual o byte dentro do bloco e o segundo bit menos significativo é usado para indicar qual o conjunto na memória tampão onde determinado dado da memória principal é guardado (Figura 5.80).

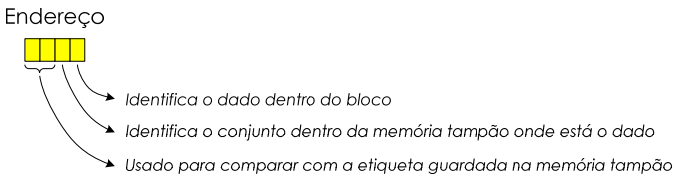


Figura 5.80 – Funções dos diferentes bits do endereço de memória principal para acesso à memória tampão.

Note-se que metade dos bytes da memória principal são sempre guardados no conjunto 0 da memória tampão enquanto a outra metade é sempre guardada no conjunto 1.

No caso do exemplo da Figura 5.79, se a palavra acedida de seguida fosse a palavra 11_{10}, que não está na memória tampão, o conteúdo do conjunto 1 seria descartado e preenchido com o bloco constituído pelos bytes do endereço 10_{10} e 11_{10} (Figura 5.81). Note-se que este bloco vai para o conjunto 1 da memória tampão porque o segundo bit menos significativo vale 1.

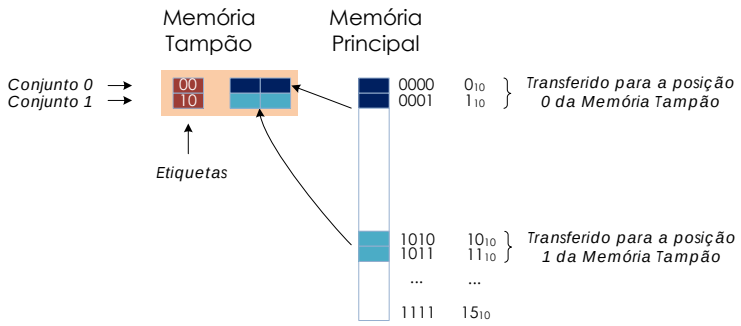


Figura 5.81 – Ilustração do preenchimento da memória tampão constituída por dois conjuntos.

Este tipo de mapeamento da memória tampão é designado por Mapeamento Directo pois cada byte da memória principal vai sempre para o mesmo sítio na memória tampão. A Figura 5.82 mostra o circuito de aceso a uma memória tampão de mapeamento directo.

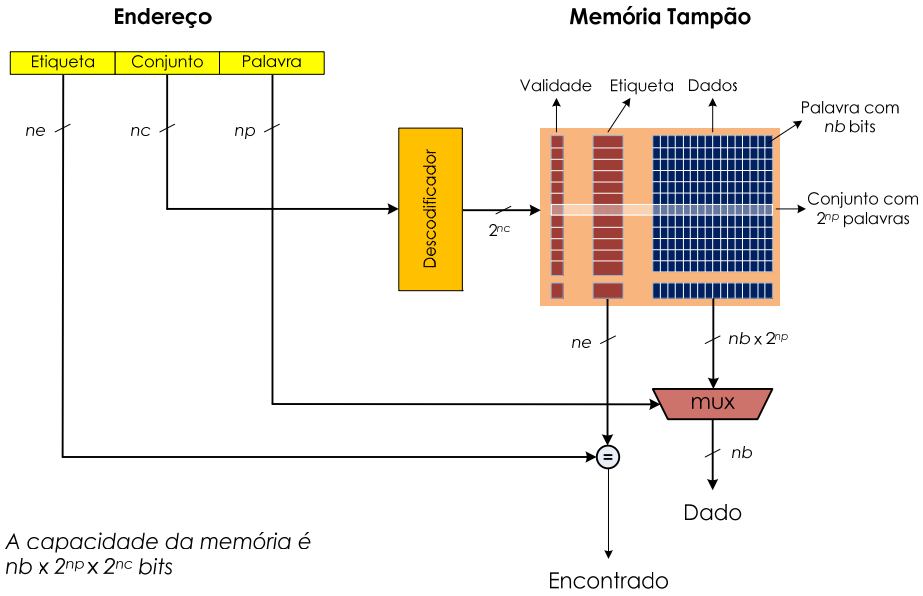


Figura 5.82 – Circuito de acesso à memória tampão com mapeamento directo.

Existe uma outra forma de organizar a memória tampão de modo a que cada palavra da memória principal possa ser armazenada em qualquer lugar da memória tampão. Esse tipo de organização chama-se de ***Completamente Associativa***. Neste caso a memória tampão é constituída por um só conjunto constituído por vários blocos que podem ser acedidos em paralelo (Figura 5.83).

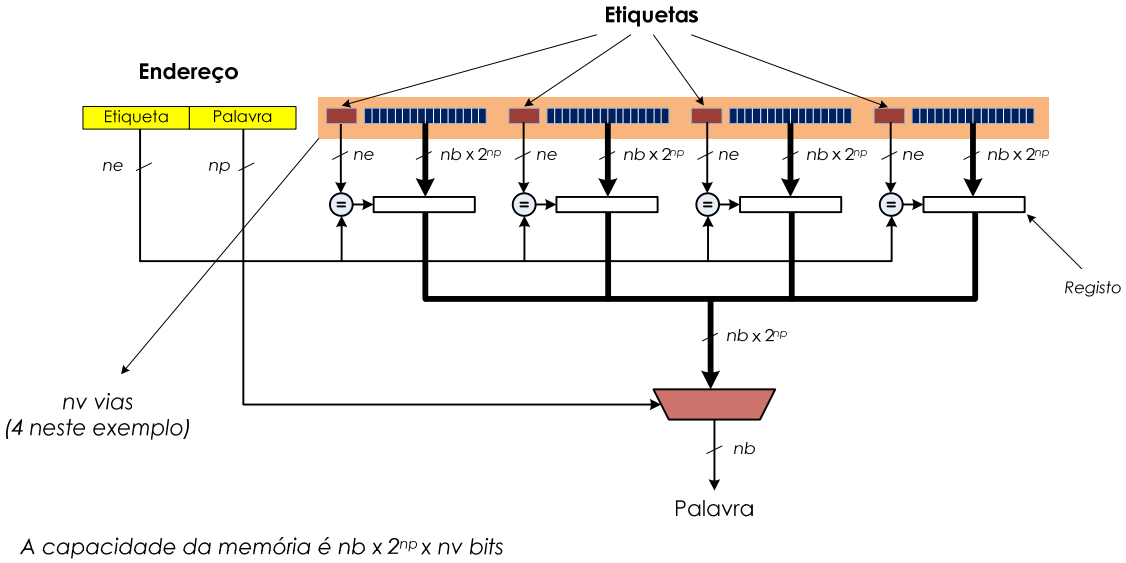


Figura 5.83 – Circuito de acesso à memória tampão completamente associativa.

Neste tipo de organização cada bloco precisa de um comparador para comparar a sua etiqueta com os bits mais significativos do endereço. Isso não é prático no caso de memórias tampão grandes. A solução consiste em combinar esta organização com a do mapeamento directo. A essa organização dá-se o nome de ***Associativa*** (Figura 5.84).

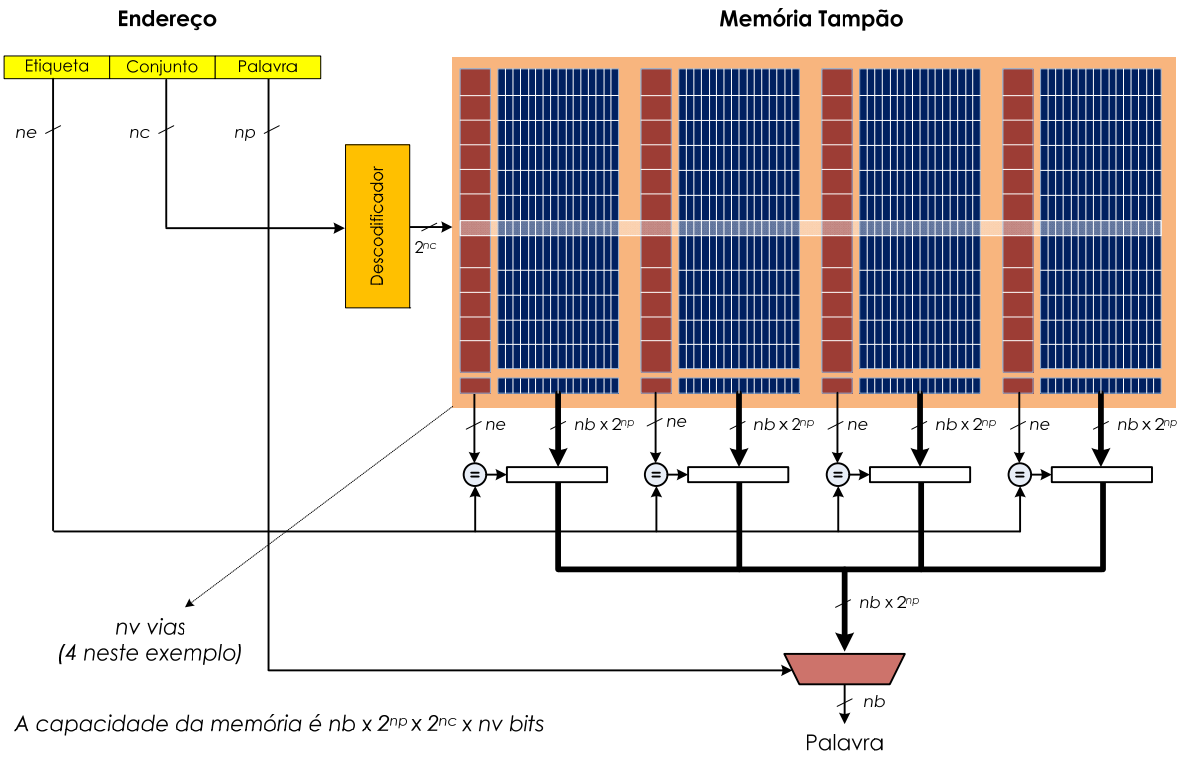


Figura 5.84 – Circuito de acesso à memória tampão associativa.

As memórias tampão associativas podem ter um número de blocos em cada conjunto diferente. No caso da Figura 5.84, por exemplo, existem 4 blocos por conjunto. Diz-se assim que é uma memória tampão associativa com *4 vias*.

O tamanho da memória tampão associativa é o produto do número palavras por bloco, do número de blocos por conjunto e do número de conjuntos. Cada um destes termos corresponde a um número de bits do endereço dado pelo logaritmo na base 2:

- Número de bits da palavra = $\log_2$ (número de palavras por bloco).
- Número de bits do índice de conjunto = $\log_2$ (número de conjuntos).
- Número de bits da etiqueta = Número de bits do Endereço – Número de bits da palavra – Número de bits do índice de conjunto.

5.5.3 Políticas de Substituição e Escrita e Estratégia de Alocação

Quando o processador pretende ler da memória e os dados não estão na memória tampão há que obtê-los da memória principal e guardá-los na memória tampão. No caso da memória tampão de mapeamento directo o endereço onde esses dados são colocados depende do endereço dos dados na memória principal. Os dados da memória principal são portanto colocados sempre no mesmo sítio da memória tampão.

No caso da memória associativa a determinação da posição ocupada consiste na determinação do conjunto e da via. O conjunto, tal como na memória tampão de mapeamento directo, depende só do endereço de memória. Já a via pode ser determinada de várias formas:

- Substituição aleatória (RR) – É escolhida uma via de forma aleatória. Não necessita que seja mantido nenhum registo sobre o uso das vias. É por isso a forma mais fácil de implementar. É usada, por exemplo, nos processadores do tipo ARM.
- Substituição da posição acedida menos recentemente (LRU) – É escolhida a via que foi acedida há mais tempo. É necessário manter um registo de quando foi acedida cada via usando-se para o efeito bits extras específicos para esta função (bits de *idade*). Quanto maior o número de vias mais complexo fica o circuito para determinar a via menos usada.
- Substituição da posição mais recentemente acedida (MRU) – É escolhida a via que foi acedida mais recentemente. Esta forma de escolha de via é vantajosa em casos em que os dados que são mais prováveis de serem necessários são os mais *antigos*. Um exemplo é quando se está a ler repetidamente um ficheiro de forma sequencial.
- Pseudo-LRU (PLRU) – É um algoritmo que quase sempre determina uma das vias menos recentemente utilizadas. Pode ser implementado com um bit que indica a via mais recentemente utilizada e depois escolhendo uma das vias adjacentes. Usado, por exemplo, no Intel Pentium 4 e no Freescale PowerPC G4 (usado pela Apple).
- Substituição da posição usada menos frequentemente (LFU) – Usa um contador para determinar quantas vezes cada via é usada.

A forma mais eficiente seria, no entanto, usar a via que contém o dado que não será necessário durante mais tempo. Este algoritmo, conhecido como algoritmo de Belady, não é no entanto realizável na prática pois não é possível saber à partida qual será esse dado. Pode ser, no entanto, usado para a avaliação a posteriori dos outros algoritmos.

Obviamente quando novos dados são colocados na memória tampão os dados que aí se encontravam são removidos.

Quando o processador pretende escrever na memória existem duas estratégias que podem ser utilizadas quando se usa uma memória tampão:

- Write Through – os dados são escritos na cache e na memória principal
    - A escrita é de apenas uma palavra e não do bloco.
- Write Back – os dados são escritos apenas na memória tampão.
    - Os blocos modificados só são escritos na memória principal quando são substituídos.
    - Para melhorar o desempenho usa-se um bit adicional por bloco que indica se o bloco foi escrito (*dirty bit*).

Na estratégia *Write Through* são usados tampões para não ter que se esperar que a memória acabe de escrever os dados. O processador escreve dados em paralelo na memória tampão e no tampão de escrita (Figura 5.85). O controlador de memória escreve o conteúdo do tampão de escrita na memória principal.

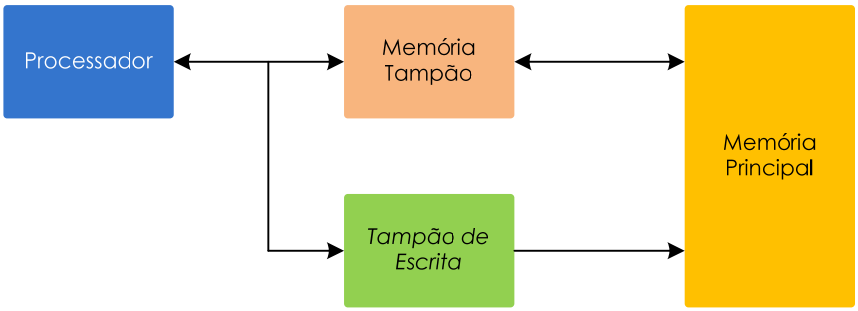


Figura 5.85 – Ilustração do uso de um tampão de escrita.

O tampão de escrita é normalmente do tipo FIFO (*first in – frist out*) e costuma ter 4 ou 8 posições de memória. Esta estratégia funciona bem se a frequência de escrita for muito menor do que o inverso do tempo de ciclo da memória principal, caso contrário o tampão de escrita enche. Neste caso é conveniente o uso de uma memória tampão de nível 2 (Figura 5.86).

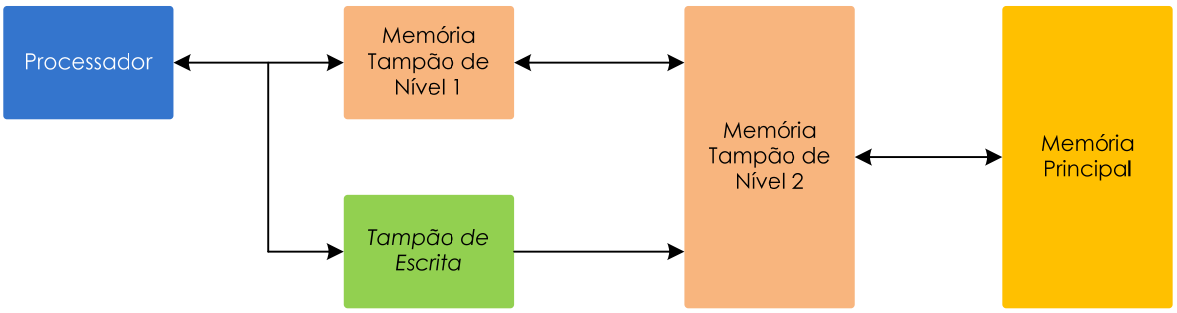


Figura 5.86 – Ilustração do uso de uma memória tampão de nível 2.

A estratégia *Write Through* tem a vantagens de que a memória principal está sempre actualizada. Por outro lado, na estratégia de *Write Back*, não se repetem escritas para a mesma posição da memória principal.

Há também duas estratégias que se podem seguir quando a palavra que se quer escrever não está na memória tampão:

- Write Allocate – Transfere-se o bloco para a memória tampão e depois escreve-se aí a nova palavra.
- Write Not-Allocate – Escreve-se apenas na memória principal. Não se transfere o bloco para a memória tampão.

Apesar de se poder utilizar a estratégia de alocação independentemente da política de escrita o que faz sentido são as combinações:

- Write through/write not allocate.
- Write back/write allocate.

5.5.4 Exemplo

Considere-se uma memória tampão de dados de nível 1 (L1) do processador AMD Athlon (arquitectura K7), com uma capacidade de 64 KB, associativa de duas vias, e com blocos de 64 bytes. A memória utiliza as estratégias *write-back* e *write-allocate* e o processador tem um comprimento de palavra de 32 bits e dispõe de 32 bits de endereço, endereçando individualmente o byte.

Na memória associativa com 2 vias existem 512 conjuntos de 64 bytes por via cada perfazendo os 64 KB da memória tampão. São portanto precisos 6 bits para endereçar a palavra dentro de conjunto ($2^6 = 64$) e 9 bits para endereçar o conjunto ($2^9 = 512$). Sobram assim 17 bits para a etiqueta ($32 - 6 - 9 = 17$).

A Figura 5.87 mostra a organização desta memória.

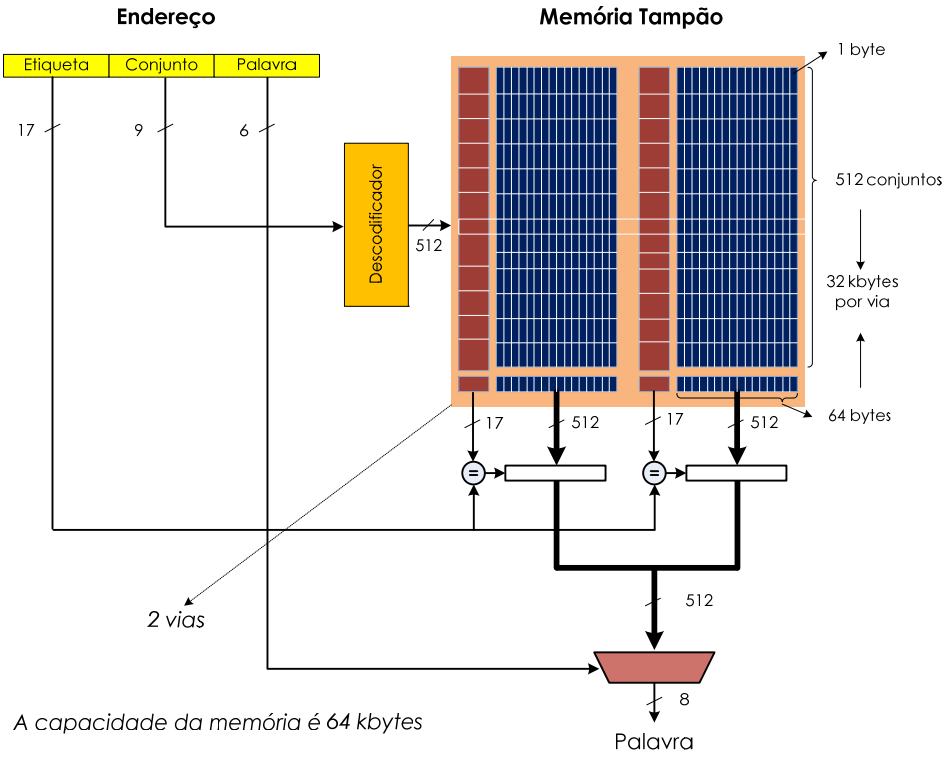


Figura 5.87 – Exemplo de organização da memória tampão de um processador AMD Athlon (arquitectura K7).

Além dos bits de armazenamento propriamente ditos existem vários bits de controlo. Esses bits são usados para as etiquetas (17 por conjunto e por via), para registo da validade dos dados armazenados (1 por conjunto por via) e para indicação de qual das vias foi usada há mais tempo. A estratégia usualmente adoptada em memórias associativas com duas vias é a técnica *least recently used* (LRU) em que a escolha da via a usar para a escrita de um novo bloco é aquela que foi usada há mais tempo. Para implementar essa estratégia basta um bit por conjunto e por via que indica se dada via é ou não a que foi usada há mais tempo. Assim que há um acesso a um bloco de memória o bit LRU desse conjunto e via é colocada a 0 e o bit correspondente da outra via é colocado a 1.

O número de bits de controlo necessários é portanto

$$(17+1+1)\times 2\times 512 = 19\times 2\times 512 = 19456\,. \tag{25}$$

Como os bits de armazenamento são 524288 (2 x 512 x 64 x 8) o número total de bits da memória tampão é 543744.

Imagine-se agora o seguinte código a executar no computador:

```
static int X[32768]; // Inteiro representado por 32 bits
register int k, sum1=0, sum2=0, sum3=0;
main()
{
        for (k = 0; k < 32768; k++) sum1 = sum1 + X[k];
        for (k = 32766; k >=0; k = k-2) sum2 = sum2 + X[k];
        for (k = 0; k < 32768; k = k+4) sum3 = sum3 + X[k];
}
```

Qual a taxa de faltas para este código?

O vector *X* ocupa 128 KB já que tem 32 x 1024 inteiros que por sua vez ocupam 4 bytes cada.

No inicio a memória tampão está vazia. No primeiro ciclo são efectuadas 32 k leituras da memória, no entanto, não há 32 k faltas pois de cada vez que há uma leitura são transferidos 64 bytes da memória. Por exemplo, quando se tenta ler o primeiro elemento do vector (*X*[0]) são imediatamente transferidos para a memória tampão 64 bytes, ou seja, 16 elementos do vector (4 bytes por elemento). Isto significa que só 1 em cada 16 acessos é que resulta em falta – há uma falta ao aceder o elemento 0 mas não há nenhuma falta ao aceder aos elementos 1 a 15. Volta no entanto a haver uma falta quando se tenta aceder o elemento 16 do vector.

Assim sendo, no primeiro ciclo, acontecem 32768 (32 k) acessos mas só 2048 faltas (32768/16). No fim do primeiro ciclo está armazenado na memória tampão a última metade do vector *X*.

No segundo ciclo os elementos do vector são lidos do fim para o princípio sendo o índice decrementado de duas posições em cada iteração. São assim efectuados 16384 acessos. Como a leitura do vector é feita a partir do fim, a segunda metade do vector (primeira metade lida) vai estar na memória tampão não havendo portanto faltas (8192 acessos sem faltas). Quando o ciclo chegar a meio os elementos do vector já não estarão na memória tampão. Devido à transferência de 16 elementos consecutivos de cada vez a leitura dos restantes 8192 elementos resultam em faltas a 1 em cada 8 acessos. Acontecem portanto 1024 faltas (8192/8).

O fim do 2º ciclo a primeira metade do vector está na memória tampão.

No terceiro ciclo o índice é incrementado de 4 unidade sendo a leitura do vector feita do inicio até ao fim (8192 acessos). Da mesma forma que no caso do 2º ciclo, a primeira metade das leituras (4096

acessos) não originará nenhuma falta. Na segunda metade, devido à transferência de blocos de 16 elementos de cada vez, haverá 1 em 4 faltas, ou seja, 1024 faltas ao todo.

Resumindo tem-se o número de acessos e de faltas em cada ciclo indicado na Tabela 5.3.

Tabela 5.3 – Resumo do número de acessos e faltas na execução do código do exemplo dados.

Ciclo	Código	Acessos	Faltas
Primeiro	for (k = 0; k < 32768; k++) sum1 = sum1 + X[k];	32768	2048
Segundo	for (k = 32766; k >=0; k = k-2) sum2 = sum2 + X[k];	16384	1024
Terceiro	for (k = 0; k < 32768; k = k+4) sum3 = sum3 + X[k];	8162	1024
Total		**57344**	**4096**

A taxa de faltas é portanto 7,1 % (4096 / 57344).

6. BARRAMENTOS

A principal função de um computador é processar informação, o que pode ser feito com uma ou mais unidades de processamento (CPU). A informação a ser processada e o resultado do processamento necessitam de ser transferidos de e para fora do processador. Essa informação pode estar armazenada num dispositivo de memória ou então transferida para um outro dispositivo interno ou externo ao computador. É necessário portanto um sistema capaz de proceder ao transporte dessa informação. A um sistema desse tipo utilizado num computador dá-se o nome de "barramento".

O objectivo último de um barramento é o de transmitir a informação o mais rapidamente possível e com o menor custo associado. Existem inúmeros tipos de barramento consoante o tipo de computador e mesmo o tipo de dispositivos envolvidos (USB, ISA, PCI, AGP, MicroChannel, PXI, etc...). Cada um desses tipos de barramento tem características ditadas pela necessidade de compromisso entre velocidade, distância, flexibilidade, privacidade, preço, etc. Por outro lado, à medida que tecnologia de fabrico de dispositivos electrónicos evolui, permitindo circuitos mais pequenos e operando com sinais de maior frequência, novos tipos de barramento surgem tornando outros obsoletos. Este capítulo apresenta os conceitos fundamentais na área dos barramentos e que são comuns a todos os tipos, de modo a permitir um entendimento dos factores que estão em jogo. Não se pretende aqui descrever os diferentes tipos de barramentos existentes embora se apresentem pontualmente alguns exemplos, pois essa informação estaria rapidamente ultrapassada tendo em conta a constante evolução e o surgimento de novos tipos de barramentos que melhoram esta ou aquela característica para uma determinada aplicação.

6.1 Representação de Informação

Para que a informação seja transmitida entre diferentes componentes electrónicos, como processadores, memórias, periféricos, é necessário que ela seja representada de uma forma eléctrica. Isso é feito utilizando campos electromagnéticos gerados por cargas em movimento. Esses campos propagam-se no espaço e com eles a informação é transportada de ponto para ponto. A propagação pode ser guiada, isto é, sustentada por exemplo num cabo de cobre ou numa fibra óptica, ou pode ser em espaço livre, usando ondas electromagnéticas no domínio óptico ou das radiofrequências.

No caso concreto dos computadores, a propagação entre componentes como o processador e a memória é efectuada, hoje em dia, de forma guiada tendo em conta que os dispositivos interligados se encontram próximos uns dos outros. Assim sendo não se justifica a utilização da propagação em espaço livre que é mais complexa, e consequentemente mais cara, e que não permite transmitir a

informação de forma tão rápida como a propagação guiada[16]. A rádio propagação é no entanto cada vez mais utilizada em computadores para ligação dos seus periféricos como seja os teclados e ratos com o objectivo de minimizar os inconvenientes da utilização de cabos. Isso é possível hoje em dia dado que a quantidade de informação transmitida entre esses periféricos é baixa. No entanto é já possível antever num futuro próximo a utilização de rádio propagação para ligação de periféricos como monitores e unidades de armazenamento externas onde a quantidade de informação transmitida é muito maior.

Controlando as características dos campos electromagnéticos de acordo com a informação que se deseja transmitir é possível efectuar a comunicação entre dois pontos. Na área das telecomunicações isto designa-se tradicionalmente por modulação. Esquemas mais ou menos complexos existem dependendo do meio utilizado. Em propagação guiada de curto alcance utiliza-se o esquema mais simples que consiste em representar a informação de forma binária, isto é, unicamente com dois símbolos (designados tradicionalmente por "0" e "1") e modificar o valor de uma tensão eléctrica constante consoante o símbolo transmitido.

A informação a transmitir entre dois ou mais componentes de um computador divide-se normalmente, em três componentes:

- **Controlo** – Informação necessária para controlar o fluxo de informação entre os dispositivos e que pode ser um sinal de relógio, por exemplo, que permite que todos os dispositivos ligados ao barramento funcionem de forma síncrona. Outro exemplo de sinal de controlo é por exemplo a indicação se é pretendido ler ou escrever dados num determinado dispositivo, como a memória.
- **Endereços** – Informação que identifica o destinatário (e ás vezes também a emissor) de certa informação que está presente no barramento.
- **Dados** – Os dados que efectivamente se querem enviar a um ou mais dispositivos.

Estas três componentes podem partilhar o mesmo canal físico, como acontece em barramentos série, em que a informação é transmitida de forma sequencial, ou usar canais físicos distintos como acontece em barramentos paralelos.

A Figura 6.1 ilustra um barramento típico entre um processador e uma memória. De forma a tornar a representação mais compacta as linhas de dados são agrupadas numa só seta. O mesmo acontece com as linhas de endereço.

[16] Isto passa-se com a tecnologia actual. Não quer dizer que no futuro, necessariamente longínquo, a comunicação entre um processador e uma memória não seja feita sem fios. Actualmente isso não se justifica.

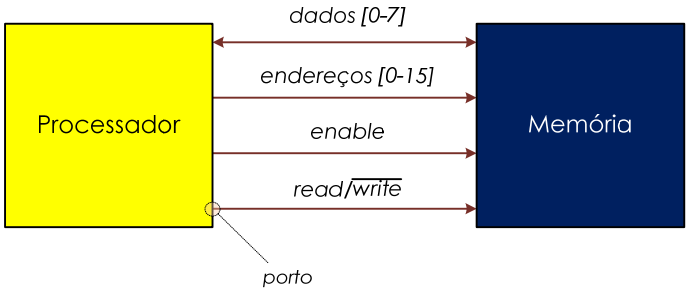


Figura 6.1 – Ilustração de um barramento típico entre um processador e uma memória.

Designa-se por "porto" um terminal de entrada/saída de um dispositivo.

Durante a comunicação entre dispositivos os diversos sinais assumem valores que variam no tempo. De modo a ilustrar a correspondência que há entre eles bem como indicar os intervalos de tempo mínimos e máximos que devem existir entre as suas transições, utilizam-se diagramas temporais como, por exemplo, o da Figura 6.2.

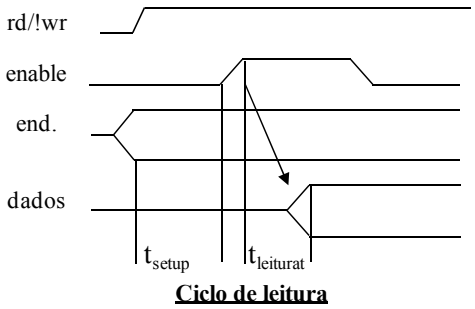


Figura 6.2 – Ilustração da variação temporal típica dos sinais numa operação de leitura.

Nesse diagrama está indicado que entre os endereços serem colocados nas linhas de barramento (linhas "endereços") e o sinal "enable" deixar de estar inactivo (nível baixo), deve existir um intervalo de tempo mínimo dado por t_{setup}. Da mesma forma, é indicado que desde que o sinal "enable" fica activo (nível alto) até que os dados lidos do dispositivo estão disponíveis nas linhas do barramento, deve existir um intervalo de tempo mínimo de $t_{leitura}$.

6.2 Interface com o Processador

Como se viu anteriormente a principal função de um processador é efectuar cálculos matemáticos e o seu funcionamento consiste em ler da memória a instrução a executar, ler os dados eventualmente necessários para efectuar os cálculos, efectuar os cálculos e eventualmente guardar o resultado na memória. O computador não está no entanto contido em si mesmo, isolado do mundo exterior. A origem e o destino dos dados podem ser externos ao computador. É necessário portanto adequar o ciclo de funcionamento típico há necessidade de interagir com dados que não estão na memória do computador.

Um das formas possíveis consiste em alterar o programa executado pelo processador para periodicamente verificar se em cada periférico existem dados novos para ser lidos. Está solução, chamada de *pooling*, tem a vantagem de não requerer nenhum hardware adicional já que a alteração consiste no programa que é executado. Tem no entanto duas desvantagens importantes. Por um lado existe o tempo perdido a verificar se há dados novos no periférico. Muitas vezes não há de facto dados novos e portanto o tempo que demora a verificação é um tempo perdido. Por outro lado, tendo em conta que essa verificação só ocorre quando o programa o previr, há a possibilidade de se perderem dados que estavam disponíveis nos periféricos e que entretanto foram substituídos por outros dados antes que a execução do programa tivesse de novo chegado à parte onde é feita a verificação de dados nos periféricos.

Refira-se que a leitura e escrita de dados nos periféricos é feita da mesma forma que na memória do computador podendo o espaço de endereçamento dividido em duas partes: uma para a memória e outra para os periféricos. Esta forma de entrada/saída de dados externos designa-se por *memory-mapped I/O*. Outra alternativa consiste em usar um sinal específico do barramento para distinguir quando o acesso é à memória ou a um periférico sendo os endereços usados os mesmos nos dois casos. Esta forma é designada por *I/O-mapped*.

Existem duas alternativas ao *pooling* que não possuem as desvantagens referidas mas que requerem, no entanto, algum hardware específico: uso de interrupções e o acesso directo à memória (DMA – *direct memory access*).

6.2.1 Interrupções

O uso de interrupções consiste em usar um sinal específico para avisar o processador que existem dados no periférico. A unidade de controlo do processador tem portanto que estar preparada para alterar o seu funcionamento normal quando verificar que esse sinal está activo. Isso consiste em armazenar num registo especial o valor actual do *program counter* (PC) e carregar no PC um endereço onde, na memória, se encontra o início de um sub-programa (rotina de interrupção) que é responsável por ler os dados do periférico e armazená-los na memória efectuando, eventualmente, algum processamento com eles. Findo esse sub-programa é carregado no PC o valor que ele continha antes de ocorrer a interrupção e a execução do programa principal continua normalmente.

Note-se que em geral é da responsabilidade da rotina de interrupção não alterar o valor dos registos do processador pois podiam estar a ser utilizados pelo programa principal ou, caso estes sejam alterados, repor os valores originais antes de chegar ao fim.

Existem duas variantes quanto à determinação do endereço de memória onde se encontra a rotina de interrupção (*interrupt address vector*). Esse valor pode ser fixo para um dado processador (*fixed interrupt*) ou então pode ser fornecido pelo periférico (*vectored interrupt*). Esta segunda alternativa tem a vantagem de possibilitar que existam diferentes rotinas de interrupção para cada periférico sendo portanto normalmente utilizada quando existem múltiplos periféricos ligados ao processador. A segunda alternativa tem, no entanto, a desvantagem de que o programa tem que saber à priori quais

os endereços de cada periférico existente. Uma solução de compromisso consiste em usar uma tabela com os endereços de memória das rotinas de interrupção e que é indexada pelo periférico.

A Figura 6.3 mostra a lógica de interrupções de um microcontrolador da Microchip Technology (modelo PIC18F4550). Observa-se que as interrupções têm um endereço fixo mas que existem dois níveis de prioridade diferentes que pode ser usados. Existem diversos sinais de controlo (*flags*) que permitem seleccionar quais os periféricos que podem provocar uma interrupção e qual a prioridade dessa interrupção. A *flag* GIE/GIEH, por exemplo, permite desabilitar por completo todas as interrupções.

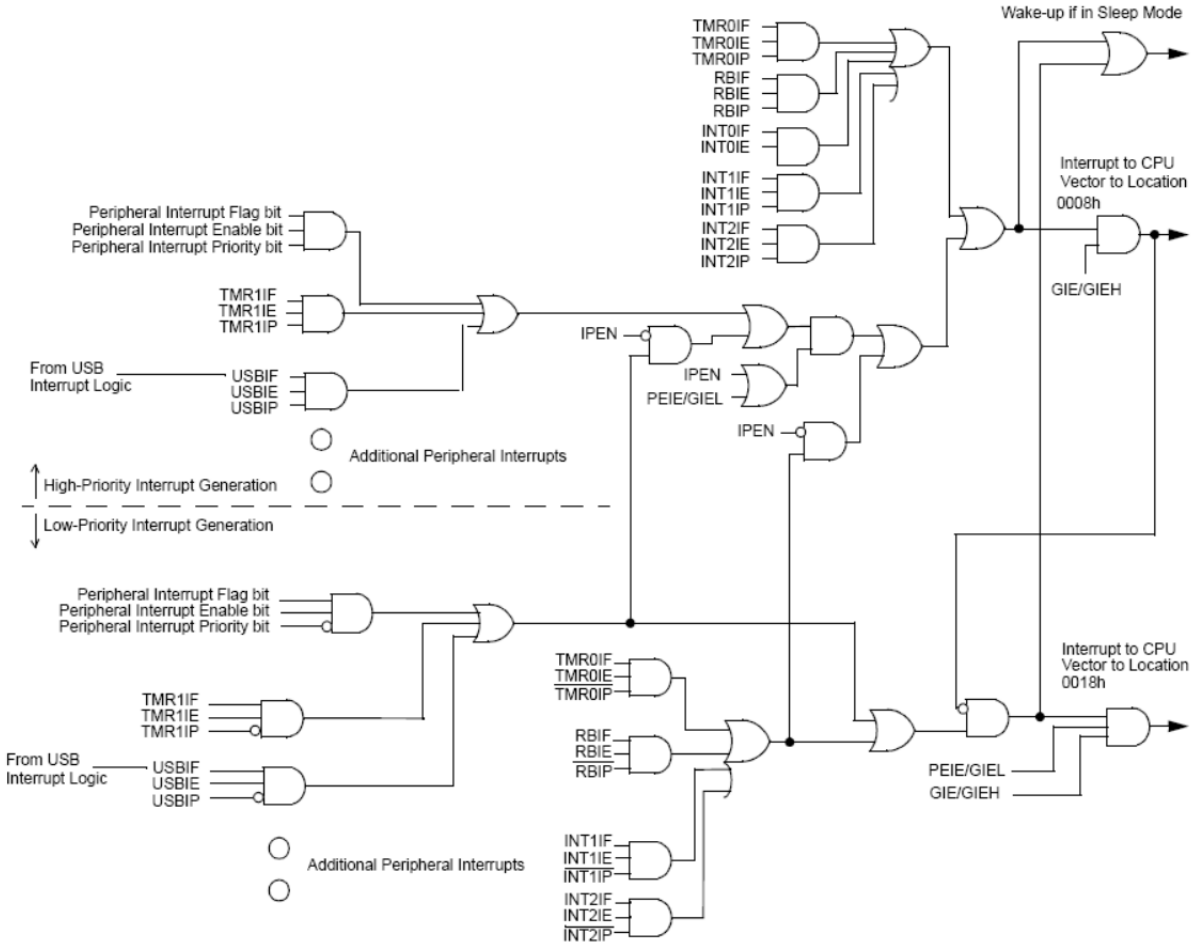
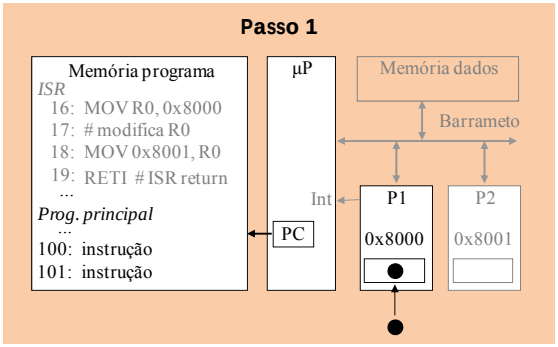
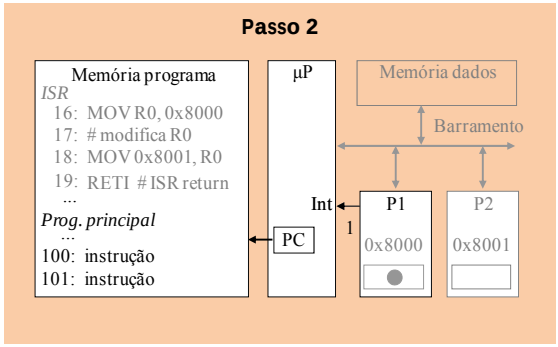


Figura 6.3 – Lógica que gere as interrupções num microcontrolador PIC18F4550 da Microchip. Retirado da sua folha de especificações. Copyright Microchip Technology Inc.

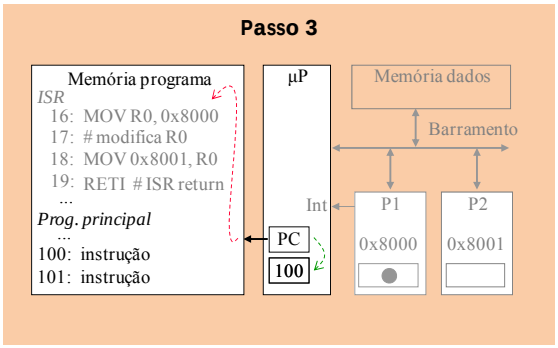
A Figura 6.4 mostra um exemplo do atendimento de uma interrupção com endereço fixo. Neste caso a rotina de interrupção lê um dado de um periférico (P1), efectua alguns cálculos e coloca o resultado no periférico P2. O endereço da rotina de interrupção é fixo e vale 16 neste exemplo.



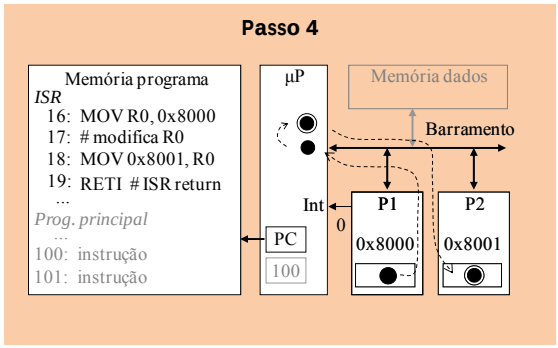
Enquando o processador está a executar o programa principal, entram dados no periférico P1 para o registo com o endereço 0x8000.



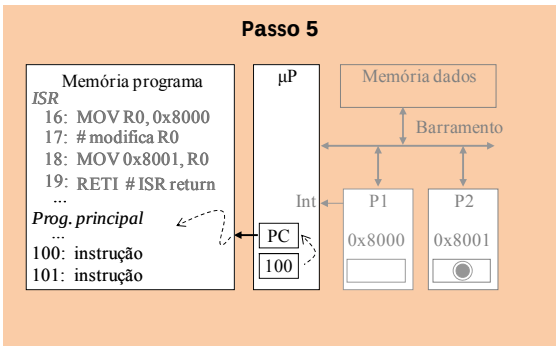
O periférico P1 activa o sinal int para indicar ao processador que tem dados novos.



O processador, depois de completar a instrução no endereço 99, verifica que int está activado e guarda o valor do PC (100) num registo especial e coloca o endereço 16 (endereço da rotina de atendimento de interrupção) no PC.



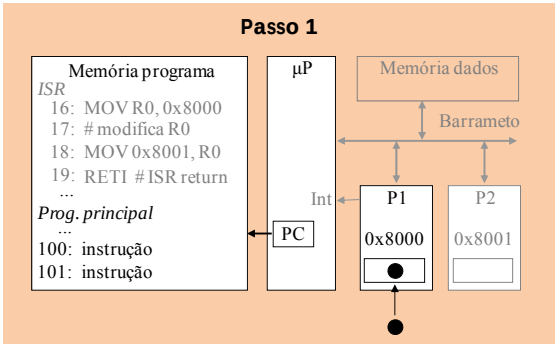
A rotina de interrupção lê os dados do registo 0x8000 do periférico P1 e escreve o resultado no registo 0x8001 de P2. Depois da leitura do valor do registo de P1 este desactiva o sinal int.



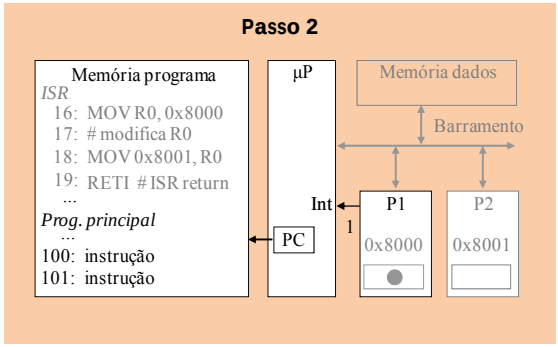
O registo PC é reposto no seu valor inicial (100) e o processador retoma a execução do programa principal.

Figura 6.4 – Exemplo do atendimento de uma interrupção com endereço fixo.

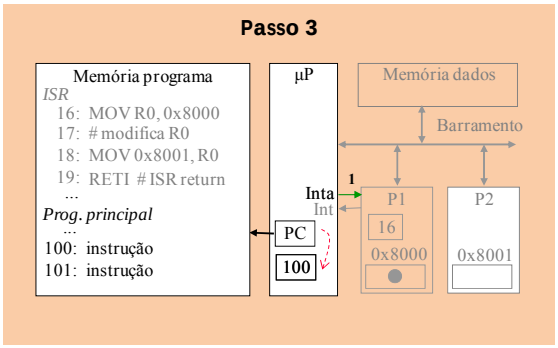
A Figura 6.9 apresenta um exemplo semelhante ao anterior mas onde é usada uma interrupção vectorizada.



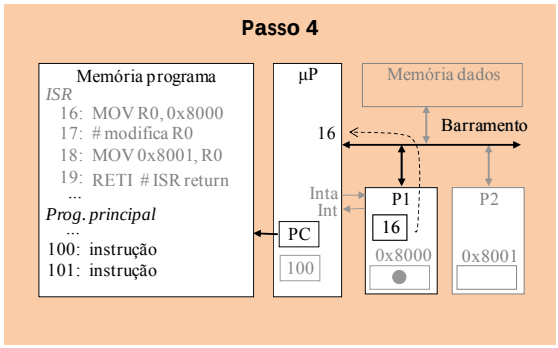
Enquando o processador está a executar o programa principal, entram dados no periférico P1 para o registo com o endereço 0x8000.



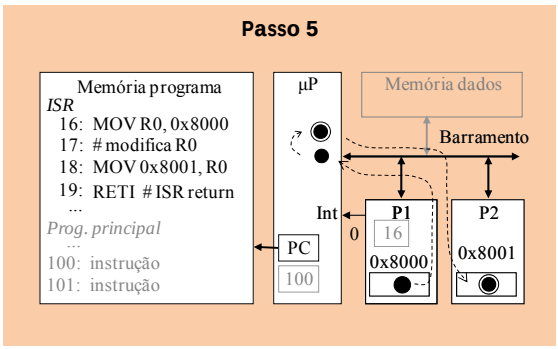
O periférico P1 activa o sinal int para indicar ao processador que tem dados novos.



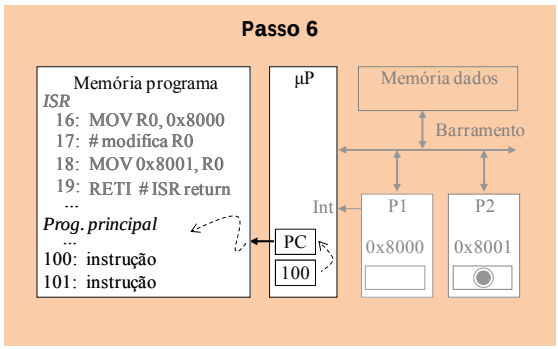
O processador, depois de completar a instrução no endereço 99, verifica que int está activado e guarda o valor do PC (100) num registo especial e activa o sinal inta.



O periférico P1 detecta que o sinal inta está activo e coloca o endereço da rotina de interrupção (16) no barramento de dados.



O processador salta para o endereço presente no barramento (16) onde se encontra a rotina de atendimento de interrupção que lê os dados do registo 0x8000 de P1, modifica-os e escreve o resultado no registo 0x8001 de P2. Depois da leitura de P1 este desactiva o sinal int.



O registo PC é reposto no seu valor inicial (100) e o processador retoma a execução do programa principal.

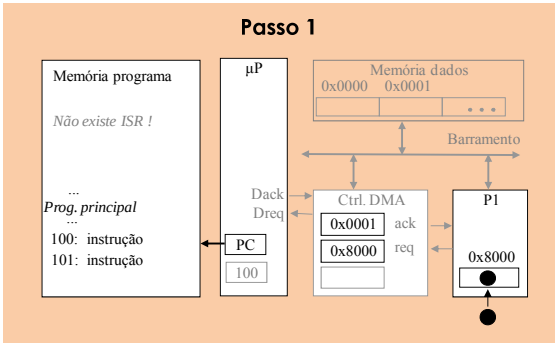
Figura 6.5 – Exemplo do atendimento de uma interrupção vectorizada.

Para além das interrupções provocadas por periféricos ligados ao processador também existem interrupções despoletadas dentro do próprio processador, em particular pela sua ALU. Essas interrupções podem assinalar, por exemplo, que o resultado de um cálculo é demasiado grande tendo em conta o número de bits usados (*overflow*).

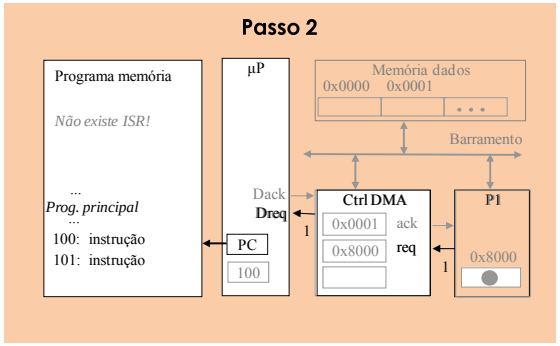
6.2.2 Acesso Directo à Memória

O acesso directo à memória (DMA – *Direct Memory Access*) permite que o conjunto de dados seja transferido de um periférico para a memória do computador sem ocupar o processador. O controlo dessa transferência é efectuado por um dispositivo dedicado – o controlador de DMA. O processador

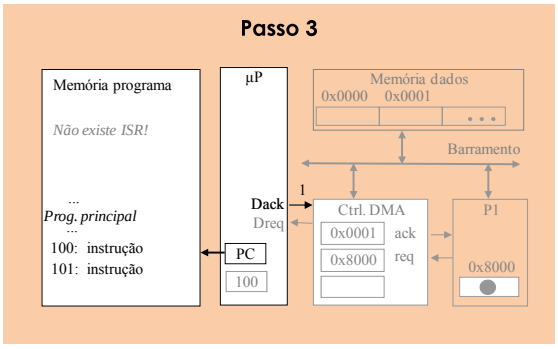
indica ao controlador de DMA a origem e o destino dos dados e a partir daí está livre para continuar a executar o programa principal. Em arquitecturas Harvard, em que os dados e as instruções estão em memórias separadas, o processador pode mesmo continuar a buscar instruções à memória enquanto a transferência de dados ocorre. Se o processador precisar de aceder à memória de dados então tem de esperar que a transferência termine.



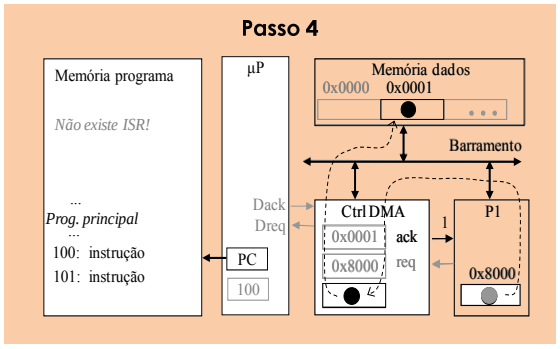
O processador já configurou o controlador de DMA e está a executar o programa principal.



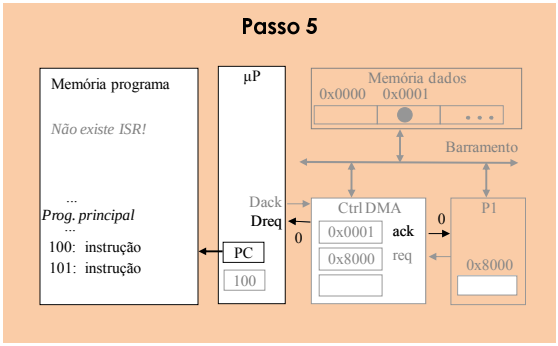
O periférico P1 activa o sinal req para solicitar o serviço do controlador de DMA que por sua vez activa Dreq para solicitar ao processador o controlo do barramento.



O processador, depois de executar a instrução 99, detecta o sinal Dreq activo, liberta o barramento, activa Dack e continua a execução do programa. Essa execução só é suspensa se o processador precisar de novo do barramento.



O controlador de DMA activa i sinal ack, lê os dados do registo 0x8000 do periférico e escreve os dados no endereço 0x0001 da memória.



O controlador de DMA desactiva Dreq e ack terminando o protocolo de aperto de mão com o periférico P1.

Figura 6.6 – Exemplo de um acesso directo à memória.

6.3 Arbitragem

Existindo diversos periféricos ligados ao mesmo barramento pode acontecer que mais do que um periférico solicite o seu uso ao mesmo tempo. Para gerir esses conflitos e estabelecer as prioridades é necessário um Árbitro.

Esse árbitro pode ser centralizado de modo que cada periférico solicite ao árbitro um determinado recurso. A Figura 6.7 ilustra esta situação em que existem 2 periféricos ligados de forma independente ao árbitro que por sua vez gere os conflitos e as prioridades requisitando o uso do barramento ao processador. O árbitro está ligado também ao barramento apenas para poder ser configurado pelo processador.

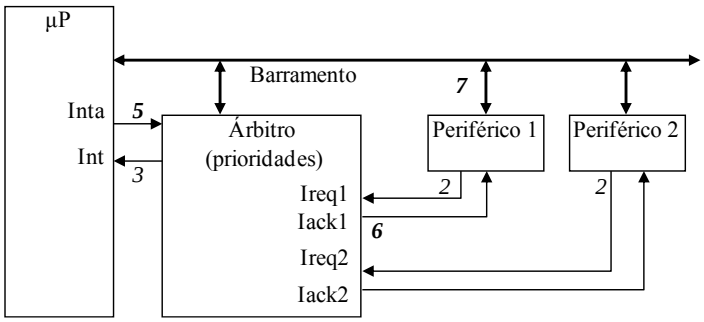


Figura 6.7 – Exemplificação da realização de arbitragem com um árbitro centralizado.

Existem duas formas de estabelecer as prioridades:

- **Prioridades Fixas** – Cada periférico tem uma ordem de prioridade pré-estabelecida. Uma ordem superior implica escolha prioritária aquando de solicitações simultâneas.
- **Prioridades Rotativas (*round-robin*)** – Prioridades alteram-se em função da história dos serviços. Isso permite uma distribuição mais equilibrada deste, especialmente entre periféricos com exigências de prioridade semelhantes.

A arbitragem pode também ser efectuada de forma descentralizada pelos próprios periféricos quer com circuitos internos quer com lógica externa.

A Figura 6.8 mostra o exemplo de uma arbitragem descentralizada do tipo *Daisy-chain*. Cada periférico tem sinais de entrada e de saída para requisição do serviço (*req*) e para confirmação da resposta (*ack*). Os periféricos estão ligados em cadeia ao processador sendo que o sinal *req* flui do periférico mais afastado para o mais próximo do processador enquanto o sinal *ack* flui do processador para o periférico mais afastado.

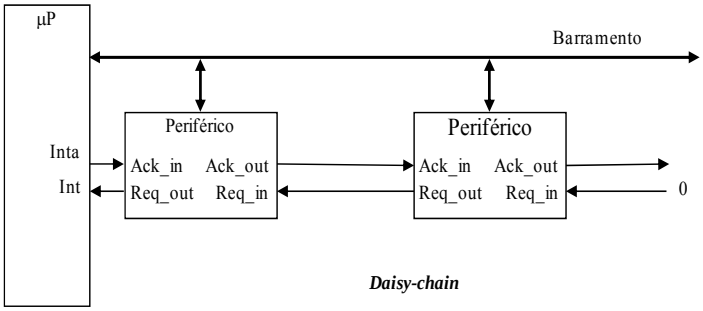


Figura 6.8 – Exemplo de uma arbitragem descentralizada do tipo Daisy-chain.

O periférico ligado directamente ao processador é o que tem maior prioridade pois tem a capacidade de bloquear os pedidos de serviço dos outros periféricos. Quando um dado periférico recebe um pedido de um outro periférico à sua direita, tendo em conta a ordem indicada na Figura 6.8, através da activação do sinal *req_in*, activa o seu sinal *req_out* só se ele próprio não tiver pedido já o serviço. Quando o processador decide atribuir o serviço (o uso do barramento) a um periférico é que activa o seu sinal de saída *Inta* que propaga-se através dos periféricos até chegar aquele que tinha um pedido pendente.

Uma desvantagem desta forma de arbitragem é que se houver uma avaria em um dos periféricos, nenhum dos outros periféricos com menor prioridade verá os seus pedidos serem atendidos por não serem transmitidos ao processador. Outra desvantagem tem a ver com o facto de que a prioridade de cada periférico é fixa só podendo ser alterada através da reordenação das ligações entre eles.

6.4 Realização Física

6.4.1 Conceitos de Linhas de Transmissão

Os sinais electromagnéticos não se propagam nos condutores que constituem um barramento de forma instantânea. Uma variação de tensão provocada por um dispositivo num dos extremos da linha demora algum tempo a se reflectir numa variação de tensão no outro extremo da linha. Para além disso, a velocidade de propagação depende da frequência do sinal. Uma sequência de 0s e 1s constitui um sinal rectangular que por sua vez pode ser descrito por uma soma de sinusóides com diferentes frequências, amplitudes e fases iniciais. A propagação dessas sinusóides de diferentes frequências com velocidades diferentes leva a uma deformação do sinal eléctrico.

Quando o tempo de propagação das ondas electromagnéticas que constituem os sinais é da ordem de grandeza do tempo de transição desses sinais deixa de se poder considerar a transmissão instantânea como se faz normalmente com sinais de baixa frequência.

Os fenómenos físicos que têm lugar nas linhas de transmissão podem ser calculados de forma rigorosa usando as equações de Maxwell que descrevem de forma geral o comportamento do campo electromagnético. O resultado da aplicação dessas equações a uma linha de transmissão constituída por dois condutores leva a que essa seja descrita por uma sucessão de secções infinitesimais cujo comportamento pode ser equiparado ao de um circuito eléctrico igual ao representado na Figura 6.9.

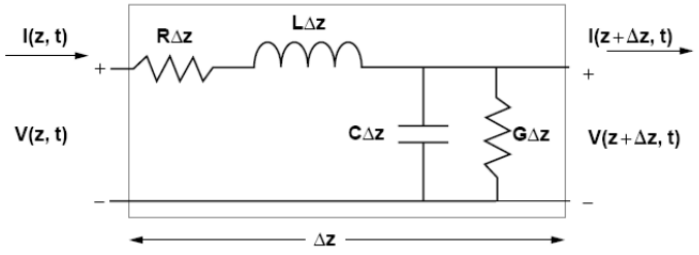


Figura 6.9 – Circuito eléctrico equivalente de uma secção infinitesimal de uma linha de transmissão.

Os 4 componentes representam 4 fenómenos físicos que estão presentes na linha de transmissão:

- Resistência R – Resistência eléctrica dos fios condutores.
- Condutância G – Condutância do meio entre os condutores.
- Indutância L – Auto-indutância devido ao campo magnético que envolve os condutores.
- Capacidade C – Capacidade entre os dois condutores devido à carga presente neles.

A tensão e a corrente na saída de cada secção de linha dependem da tensão e corrente na sua entrada e dos 4 parâmetros constitutivos da linha. Estes 4 parâmetros são definidos por unidade de comprimento.

$$\begin{aligned} v(z,t) &= R\cdot\Delta z\cdot i(z,t) + L\cdot\Delta z\cdot\frac{\partial i(z,t)}{\partial t} + v(z+\Delta z,t) \\ i(z,t) &= G\cdot\Delta z\cdot v(z+\Delta z,t) + C\cdot\Delta z\cdot\frac{\partial v(z+\Delta z,t)}{\partial t} + i(z+\Delta z,t) \end{aligned} \quad . \tag{26}$$

Estas equações podem ser reescritas de forma a evidenciar a variação de tensão e corrente por unidade de comprimento ao longo de uma secção de linha:

$$\begin{aligned} \frac{v(z+\Delta z,t)-v(z,t)}{\Delta z} &= -R\cdot i(z,t) - L\cdot\frac{\partial i(z,t)}{\partial t} \\ \frac{i(z+\Delta z,t)-i(z,t)}{\Delta z} &= G\cdot v(z+\Delta z,t) + C\cdot\frac{\partial v(z+\Delta z,t)}{\partial t} \end{aligned} \quad . \tag{27}$$

No limite em que o comprimento do troço Δz tende para zero tem-se a derivada espacial da tensão e da corrente:

$$\begin{aligned} \lim_{\Delta z\to 0}\frac{v(z+\Delta z,t)-v(z,t)}{\Delta z} &= \frac{\partial v(z,t)}{\partial z} \\ \lim_{\Delta z\to 0}\frac{i(z+\Delta z,t)-i(z,t)}{\Delta z} &= \frac{\partial i(z,t)}{\partial z} \end{aligned} \quad . \tag{28}$$

Obtêm-se assim as equações diferencias que definem a relação entre tensão e corrente ao longo da linha.

$$\begin{aligned} \frac{\partial v(z,t)}{\partial z} &= -R\cdot i(z,t) - L\cdot\frac{\partial i(z,t)}{\partial t} \\ \frac{\partial i(z,t)}{\partial z} &= -G\cdot v(z,t) - C\cdot\frac{\partial v(z,t)}{\partial t} \end{aligned} \quad . \tag{29}$$

Em regime sinusoidal a tensão e a corrente podem ser descritas por amplitudes complexas[17],

$$\begin{aligned}\overline{V} &= Ve^{j\omega t}\\ \bar{I} &= Ie^{j\omega t}\end{aligned}. \tag{30}$$

Reescrevendo (30) usando as amplitudes complexas leva a

$$\begin{aligned}\frac{\partial \overline{V}(z)}{\partial z} &= -(R+j\omega L)\cdot \bar{I}(z)\\ \frac{\partial \bar{I}(z)}{\partial z} &= -(G+j\omega C)\cdot \overline{V}(z)\end{aligned}. \tag{31}$$

Aplicando a derivada em ordem a *z* nos dois membros das equações permite chegar a

$$\begin{aligned}\frac{\partial^2 \overline{V}(z)}{\partial z^2} &= \bar{\gamma}^2\cdot \overline{V}(z)\\ \frac{\partial^2 \bar{I}(z)}{\partial z^2} &= \bar{\gamma}^2\cdot \bar{I}(z)\end{aligned}, \tag{32}$$

em que

$$\bar{\gamma} = \sqrt{(R+j\omega L)(G+j\omega C)}\,. \tag{33}$$

À variável γ dá-se o nome de constante de propagação. Esta constante depende dos parâmetros constitutivos da linha e da frequência angular dos sinais sinusoidais.

A função matemática que satisfaz equações diferencias de 2ª ordem como as que se tem em (32) é a função exponencial, podendo o argumento ser positivo ou negativo. A solução genérica é portanto uma combinação linear destas funções,

$$\begin{aligned}\overline{V}(z) &= \overline{V_+}e^{\bar{\gamma}z} + \overline{V_-}e^{-\bar{\gamma}z}\\ \bar{I}(z) &= \frac{\overline{V_+}}{\overline{Z_0}}e^{\bar{\gamma}z} - \frac{\overline{V_-}}{\overline{Z_0}}e^{-\bar{\gamma}z}\end{aligned}. \tag{34}$$

Esta solução pode ser interpretada como representando a soma de duas ondas que se propagam em sentidos contrários da linha. Note-se que a soma de um qualquer número de exponenciais é também solução de (32). Podem portanto existir na linha um número arbitrário de ondas a propagarem-se nos dois sentidos.

[17] Uma das formas de indicar que determinada variável representa uma amplitude complexa é o uso de uma barra horizontal por cima de uma letra maiúscula. Quando se trabalha num problema em que todas as variáveis se podem assumir como sendo complexas em geral dispensa-se o uso da barra vertical por cima da variável.

A amplitude das ondas de corrente está relacionada com a amplitude das ondas de tensão pela *Impedância Característica da Linha* dada por

$$\overline{Z_0} = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \,. \tag{35}$$

Note-se que em geral a impedância característica não é a relação entre a tensão e a corrente em cada ponto da linha mas sim a relação entre a tensão e a corrente em cada ponto da linha **para cada onda**. Tem-se portanto que em geral

$$\frac{\overline{V}(z)}{\overline{I}(z)} \neq \overline{Z_0} \,. \tag{36}$$

O único caso particular em que isso acontece é quando se tem só uma onda a propagar-se na linha. Isso verifica-se em linha infinitas ou terminadas por cargas adaptadas. Uma carga adaptada é uma carga que tem uma impedância cujo valor é o complexo conjugado da impedância característica da linha.

Quando a frequência é alta os termos *R* e *G*, que representam fenómenos físicos associados a perdas, tornam-se desprezáveis face a $j\omega L$ e $j\omega C$ respectivamente. Neste caso tem-se

$$\overline{\gamma} = j\omega\sqrt{LC} \quad \text{e} \quad \overline{Z_0} = \sqrt{\frac{L}{C}} \,, \tag{37}$$

que são a constante de propagação e a impedância característica de uma linha sem perdas.

No caso da linha sem perdas a velocidade de propagação é dada por

$$v = \frac{1}{\sqrt{LC}} \,. \tag{38}$$

A Figura 6.10 mostra o exemplo de uma placa de circuito impresso de 6 camadas. Duas dessas camadas são usada como plano de massa (M2) e como plano de alimentação (M5). As outras 4 camadas contêm as pistas que têm assim uma configuração de *microstrip* (M1 e M6) ou *stripline* (M3 e M4).

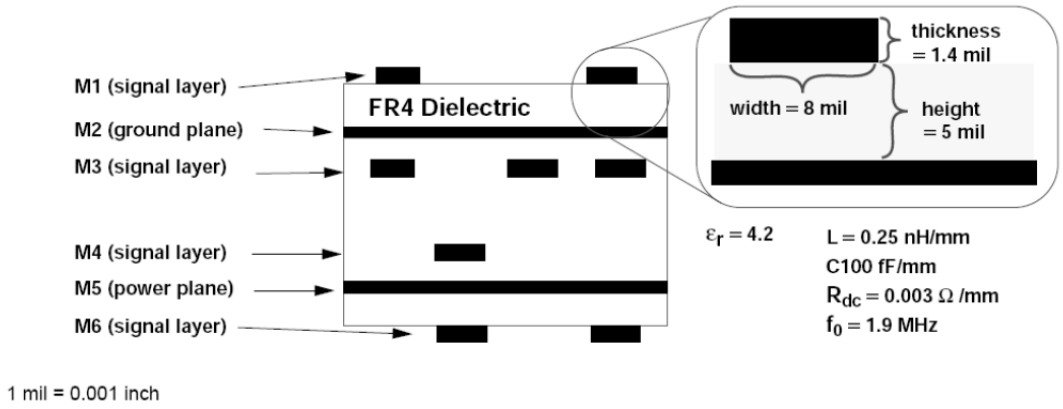


Figura 6.10 – Exemplo dos parâmetros constitutivos de uma *microstrip*.

No caso da *stripline* com as dimensões representadas na figura, a velocidade de propagação é de 2×10^8 m/s, o que corresponde a aproximadamente 66% da velocidade das ondas electromagnéticas no vácuo.

Um parâmetro mais comum de se usar quando se tratam de pistas numa placa de circuito impresso ou num circuito integrado é o inverso da velocidade (dividido por 1000) que representa o tempo que as ondas levam a percorrer 1 mm (t_p):

$$t_p = \frac{1}{v} \times \frac{1}{1000}. \tag{39}$$

No caso do exemplo anterior esse tempo é de 5 ps/mm.

Cada onda tem uma amplitude que depende das fontes de sinal e das terminações da linha. Uma terminação, por exemplo, impõe uma certa condição ao valor da tensão e/ou ao valor da corrente que deve ser satisfeita nesse ponto da linha. Se essa condição não for verificada pela onda de tensão/corrente incidente, surge uma onda reflectida de modo que a sua soma com a onda incidente verifique, no fim da linha, a condição imposta pela terminação.

Por exemplo, no caso de uma terminação em curto-circuito a condição imposta é a de que a tensão no fim da linha tem que ser zero. Se a tensão que a onda incidente tem quando chega ao fim da linha não for zero surgirá uma onda reflectida com um valor no fim da linha simétrico ao valor da onda incidente. Essa onda propagar-se-á no sentido contrário ao da onda incidente. Em geral, em cada ponto da linha a tensão resultante da soma das ondas incidente e reflectida não será zero. Isso só acontece no fim da linha e em pontos que distam um número inteiro de vezes meio comprimento de onda do fim a linha.

A onda reflectida no fim da linha e que se propaga no sentido contrário ao da onda incidente poderá eventualmente dar origem a uma nova onda reflectida no início da linha (onde a onda incidente tinha sido criada). Essas reflexões podem ocorrer sucessivamente em ambas as extremidades da linha de tal modo que o número de ondas aumenta com o tempo. A soma dessas ondas, em cada ponto da linha, tende para um valor. O diagrama desse valor (da soma vectorial das amplitudes complexas

de cada onda) em função da distância ao fim (ou ao início) da linha é chamado de diagrama de onda estacionária (Figura 6.11). É o valor presente nesse diagrama que é possível medir com um voltímetro ou visualizar num osciloscópio e não o valor da amplitude de cada onda individual.

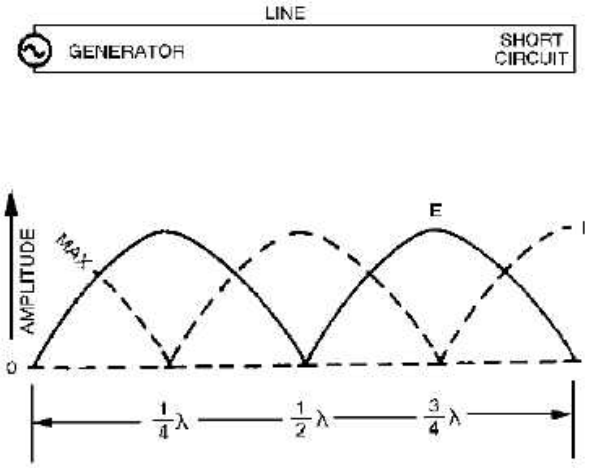


Figura 6.11 – Exemplo de um diagrama de onda estacionária de uma linha de transmissão terminada com um curto-circuito.

A impedância característica de uma linha de transmissão, quer seja um cabo ou uma pista numa placa de circuito impresso ou num circuito integrado, depende da sua geometria. A presença de curvas, cantos, entroncamentos, cruzamentos, vias, soldaduras, pinos de encapsulamento dos integrados, *wire bonds* ou *pads* no caminho de um sinal leva inevitavelmente ao surgimento de ondas reflectidas devido à desadaptação de impedância. A Figura 6.12 ilustra as ondas na interface de dois troços de linha com impedâncias diferentes.

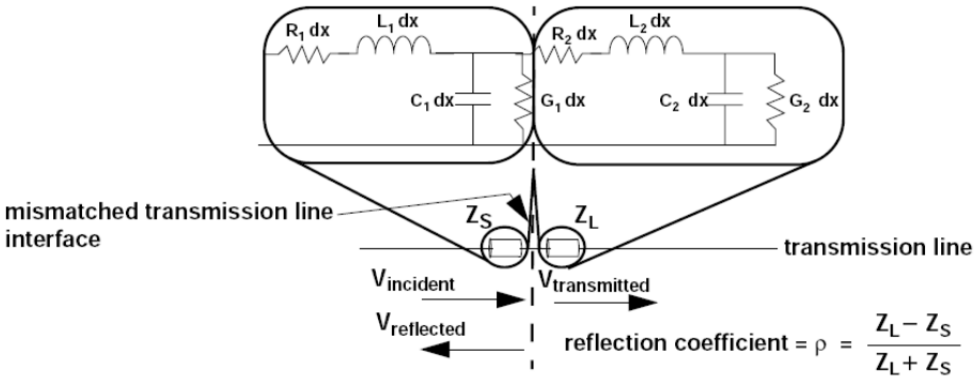


Figura 6.12 – Ilustração das ondas existentes na interface entre dois troços de linha com impedâncias diferentes.

A onda incidente vinda da esquerda dá origem a uma onda transmitida que segue para o troço da direita e a uma onde reflectida que se propaga no sentido contrário na mesma linha. A amplitude dessas ondas está relacionada com a amplitude da onda incidente através do coeficiente de transmissão ($\overline{T}$) e de reflexão ($\overline{\Gamma}$):

$$\overline{T} = \frac{\overline{V}_{transmitida}}{\overline{V}_{incidente}} \quad \text{e} \quad \overline{\Gamma} = \frac{\overline{V}_{reflectida}}{\overline{V}_{incidente}}. \tag{40}$$

Esses coeficientes dependem directamente das impedâncias características dos dois troços de linha. No caso da figura tem-se

$$\overline{T} = \frac{2\overline{Z}_S}{\overline{Z}_L + \overline{Z}_S} \quad \text{e} \quad \overline{\Gamma} = \frac{\overline{Z}_L - \overline{Z}_S}{\overline{Z}_L + \overline{Z}_S}. \tag{41}$$

A Figura 6.13 ilustra o caso de uma linha de transmissão com uma impedância característica de 100 Ω deixada em aberto (carga com impedância infinita) estimulada por um gerador de sinal com uma impedância de 50 Ω.

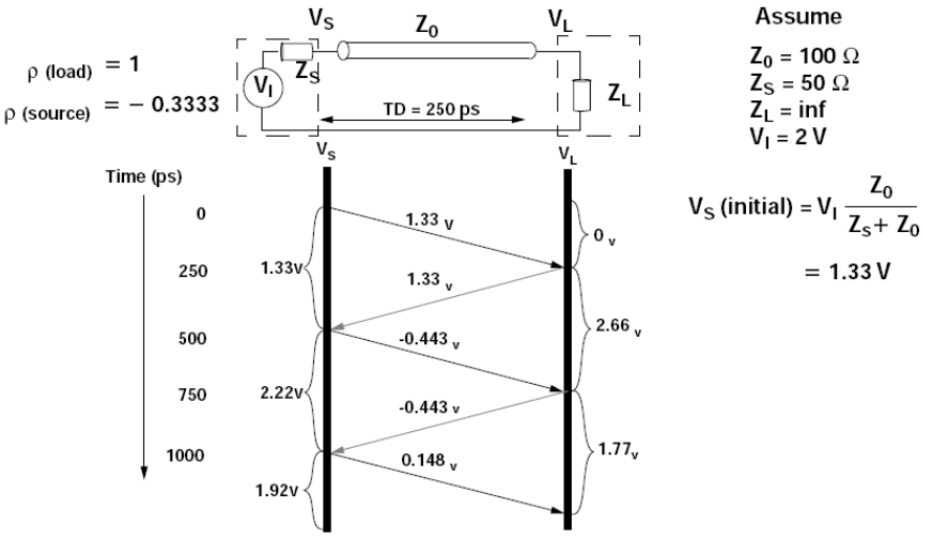


Figura 6.13 – Ilustração da evolução da tensão nos extremos de uma linha desadaptada.

O gerador de sinal impõem uma tensão de 2 V o que dá origem a uma onda com uma amplitude de 1,33 V. Esse valor obtém-se da divisão de tensão provocada pela impedância do gerador e pela impedância característica da linha:

$$\overline{V}_{incidente} = 2 \times \frac{\overline{Z}_0}{\overline{Z}_0 + \overline{Z}_S} = 2 \times \frac{100}{100 + 50} = 1{,}33 \text{ V}. \tag{42}$$

Assume-se que a fase inicial da onda incidente é nula no instante t = 0.

No instante inicial em que o gerador é ligado a tensão no extremo esquerdo da linha é 1,33 V e no extremo direito é 0 pois a onda gerada ainda não teve tempo de percorrer a linha. Só passado esse tempo (250 ps no caso do exemplo da figura), é que a onda incidente atinge o fim da linha. Essa onda não satisfaz a condição fronteira imposta pelo circuito aberto – corrente nula. A corrente da onda incidente é 13,33 mA:

$$I_{incidente} = \frac{V_{incidente}}{Z_0} = \frac{1,33}{100} = 13,33 \text{ mA} . \quad (43)$$

Nasce portanto uma onda reflectida com uma amplitude de corrente de -13,33 mA. A tensão dessa onda é de 1,33 V. Relembre-se que a relação entre a amplitude da tensão e da corrente de uma onda que viaja no sentido contrário ao convencionado como positivo é o simétrico da impedância característica (ver (34)). Não há, neste caso nenhuma onda transmitida nem nenhuma potência dissipada na carga.

A onda reflectida com 1,33 V de amplitude viaja até ao início da linha onde dá origem a uma nova onda desta vez com uma amplitude de -0.443 V calculado a partir do coeficiente de reflexão no início da linha:

$$\overline{\Gamma}_S = \frac{\overline{Z}_S - \overline{Z}_0}{\overline{Z}_S - \overline{Z}_0} = \frac{50-100}{50+100} = -0,333 . \quad (44)$$

Note-se que neste caso, devido à impedância do gerador, parte da energia da onda que atingiu o gerador (onda com amplitude 1,33 V) é dissipada no gerador. A energia restante constitui a segunda onda incidente que parte em direcção ao fim da linha no instante 500 ps.

O gerador está continuamente a injectar o sinal na linha pelo que as ondas nunca desaparecem. A partir do instante 500 ps existem 3 ondas a viajar na linha. A tensão em cada ponto da linha é a soma da tensão de cada onda. Logo a seguir ao instante 500 ps, por exemplo, a tensão no início da linha é de 2,22 V (1,33 + 1,33 – 0.443).

Vai, portanto, existir reflexões nos dois extremos da linha o que faz com que o número de ondas que viajam na linha aumente com o tempo. A tensão em cada ponto da linha vai tender para o valor 2 V, que era a tensão imposta pelo gerador, passado algum tempo como se ilustra na Figura 6.14 para o fim da linha.

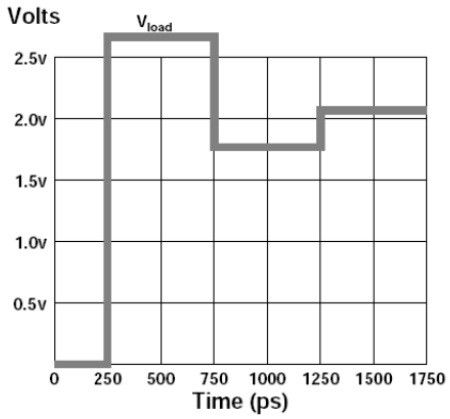


Figura 6.14 – Ilustração da evolução da tensão no fim da linha para o exemplo numérico apresentado.

Este exemplo mostra que quando há uma variação brusca de tensão num extremo de uma linha a tensão no outro extremo sofre algumas oscilações até se aproximar desse valor passado algum tempo.

Quando existem vários dispositivos ligados a uma linha de transmissão, como acontece, por exemplo, num barramento usado para ligar um processador a vários circuitos integrados de memória, existirão potencialmente várias ondas reflectidas em cada uma dessas conexões (Figura 6.15).

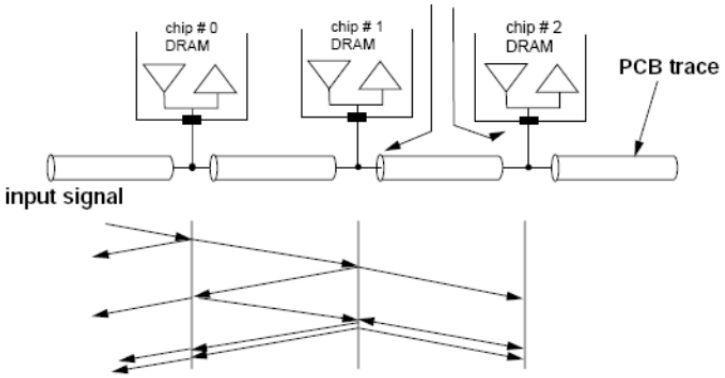


Figura 6.15 – Barramento com várias ligações.

A ligação de um dispositivo a um barramento não dá origem a uma só descontinuidade mas a várias pois é constituída por diferentes interfaces. No caso da Figura 6.16, onde se representa o barramento num sistema de memória DDR SDRAM, observa-se a interface da linha com o *socket*, do *socket* com o módulo de memória deste com os circuitos integrados com a memória DRAM.

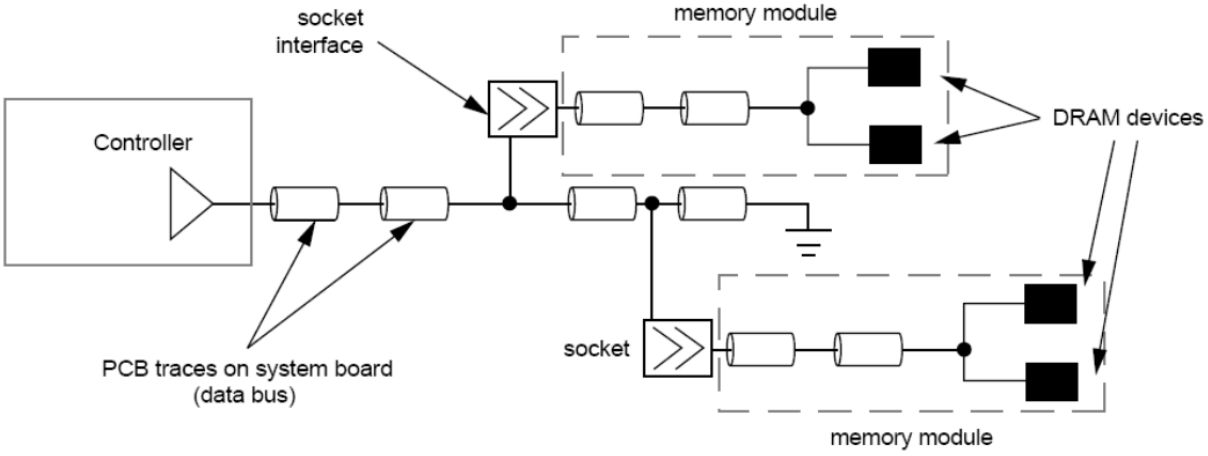


Figura 6.16 – Representação de um barramento num sistema de memória DDR SDRAM.

Outro problema que se encontra em barramentos devido à velocidade finita de propagação dos sinais tem a ver com diferentes comprimentos das pistas referentes a um conjunto de linhas que transmitem uma palavra em paralelo como o caso típico dos dados ou endereços. Esses diferentes comprimentos fazem com que os flancos dos sinais que estavam temporalmente alinhados na fonte não estejam alinhados quando chegam ao se destino.

Para minimizar esse problema tenta-se, sempre que possível ter pistas com o mesmo comprimento como se observa, por exemplo, na Figura 6.17.

Figura 6.17 – Placa de circuito impresso mostrando pista cujo comprimento foi tornado propositadamente mais longo do que seria de esperar.

6.4.2 Terminações

Uma das formas de minimizar as reflexões existentes numa linha de transmissão é o uso de dispositivos que tenham uma impedância de saída e uma impedância de entrada o mais próximo da impedância característica da linha.

Tipicamente sucede que a impedância de saída das fontes de sinal é menor do que a impedância característica da linha e que a impedância de entrada das cargas é maior do que a impedância característica da linha [7]. Uma baixa impedância de saída significa que a fonte é capaz de fornecer bastante corrente enquanto uma alta impedância de entrada significa que a carga consome pouca corrente. Isso é vantajoso, por exemplo, quando se pretende que uma determinada fonte de sinal esteja conectada a várias cargas (*fan-out*) o que é típico em circuitos digitais.

A desadaptação de impedâncias entre fonte, linha e carga tem, no entanto, a desvantagem de levar à existência de reflexões de sinal que causam diversos problemas como o disparo provocado por sinais de relógio em instantes errados, interpretação errada dos bits de dados, endereços e linhas de controlo e aumento da interferência electromagnética causada em outros circuitos adjacentes.

Existem diversas técnicas para realizar a adaptação das fontes e cargas às linhas. Essas técnicas podem classificar-se em passivas, usando resistências, condensadores e diodos, e activas onde são usados reguladores ou amplificadores operacionais.

A técnica mais simples para adaptação de uma carga a uma linha consiste na colocação de uma resistência entre a linha e a alimentação ou entre a linha e a massa, como ilustrado na Figura 6.18.

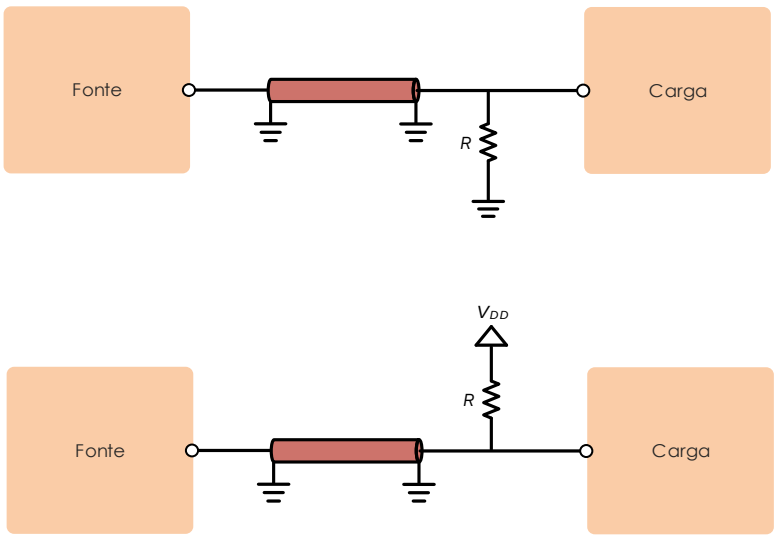


Figura 6.18 – Terminação paralela constituída por uma única resistência ligada à massa.

Este tipo de terminação é conhecida como ***terminação paralela***. A resistência deve ter um valor igual à impedância característica da linha,

$$R = Z_0 \,. \tag{45}$$

A ligação da resistência à alimentação ajuda a fonte a fornecer corrente à carga pois a tensão à entrada da carga será tipicamente inferior à tensão de alimentação o que faz com que haja uma corrente a fluir da alimentação para a carga que se soma à corrente vinda da fonte.

A Figura 6.19 ilustra a situação em que o nível lógico à saída da fonte passa de 0 para 1. Isso faz com que a tensão à sua saída passe a ser aproximadamente V_{DD}. Flui então uma corrente na direcção da carga, através da linha de transmissão. Essa corrente, que consiste num deslocamento de electrões no sentido contrário, faz com que a carga eléctrica na porta do transístor de entrada do dispositivo Carga se torne positiva. Relembre-se que a porta de um transístor MOSFET comporta-se com um condensador. O seu carregamento, neste caso, consiste em remover electrões da porta (que estão na banda de condução do metal dessa porta) o que por sua vez vai atrair electrões do substrato do transístor para perto da porta (formando um canal entre a fonte e o dreno).

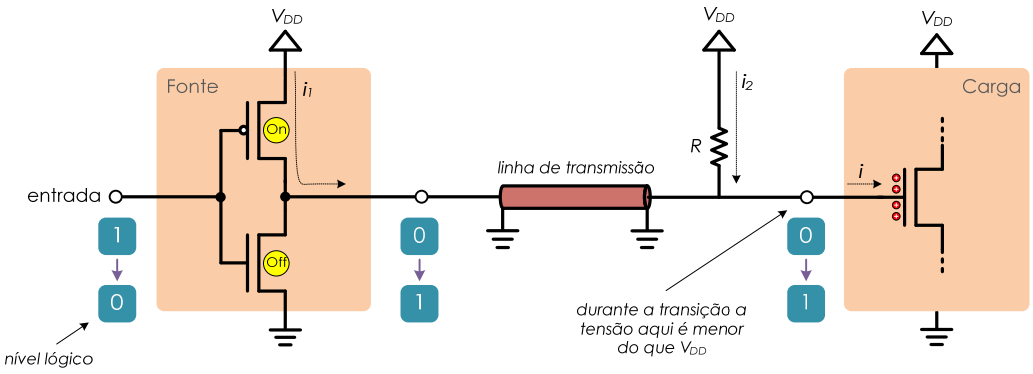


Figura 6.19 – Ilustração de uma porta NOT ligada a um MOSFET. Caso em que a saída da porta NOT está no nível alto.

Quanto maior for o fluxo de electrões para longe da porta do transístor mais depressa ele muda de estado. A existência da resistência *R*, ligada a V_{DD} na entrada da carga, faz com que haja um novo caminho para esses electrões chegarem à fonte de alimentação (maior corrente na porta do transístor) pois a tensão no terminal superior será maior do que a tensão no terminal inferior durante a transição de 0 para 1. Eventualmente o valor da carga positiva na entrada do transístor fará com que a tensão à entrada da carga seja igual a V_{DD} deixando de fluir corrente ($i = 0$) – os circuitos CMOS só consomem corrente quando estão a mudar de estado.

Por seu lado, a ligação da resistência à massa faz com que haja uma corrente que percorre essa resistência em direcção à massa, devido à maior tensão à entrada da carga em relação à massa, escoando assim parte da corrente que vem da carga. O resto da corrente flui em direcção à fonte.

A Figura 6.19 ilustra a situação em que a saída da fonte muda do nível lógico 1 para o 0. Há assim uma corrente que flui através do transístor nMOS da porta NOT em direcção à massa. Isso corresponde a um fluxo de electrões em direcção à porta do transístor de entrada da carga que vai anular a carga positiva que lá existia. A existência de uma resistência ligada à massa faz com que haja uma corrente adicional em direcção à massa o que aumenta a corrente que saí do terminal de entrada da carga. Quanto maior essa corrente mais depressa o transístor deixa de conduzir.

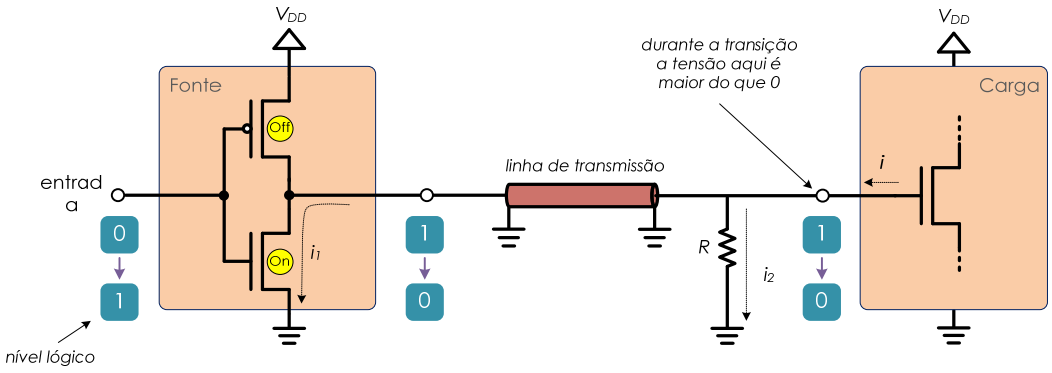


Figura 6.20 – Ilustração de uma porta NOT ligada a um MOSFET. Caso em que a saída da porta NOT está no nível alto.

A existência de uma resistência à saída da linha de transmissão ligada à massa ou à alimentação positiva, além de absorver a energia vinda da fonte de modo a evitar a existência de ondas reflectidas, também torna a mudança de estado do transístor de entrada da carga mais rápida. Para além disso aumenta-se o ***fan-out***, o seja, é possível ligar mais transístores na carga ligados a esta entrada.

Cada uma das soluções apresentadas só trás vantagens para um dos níveis lógicos. Admitindo que os dois níveis lógicos são igualmente prováveis de acontecer (ciclo de trabalho de 50%), ambas as situações são igualmente boas. Nos circuitos CMOS, no entanto, isso não é bem assim. A saída de circuitos CMOS consegue mais facilmente absorver corrente do que fornecer corrente devido às diferentes resistências dos transístores nMOS e pMOS. A maior mobilidade dos electrões quando comparada com a das lacunas faz com que os transístores nMOS tenham uma resistência entre fonte e dreno menor do que nos transístores pMOS (caso ambos tenham as mesmas dimensões). Assim sendo a ligação da resistência de terminação à alimentação positiva é a mais indicada para circuitos CMOS.

Outro efeito causado pelo uso de uma terminação paralela é a redução das margens de ruído. A margem de ruído de um circuito digital é a diferença entre o valor da saída e o valor de entrada das portas lógicas. As portas lógicas têm que produzir, no caso do nível lógico 0, uma tensão obrigatoriamente inferior a V_{OL} (***output low***) e, no caso do nível lógico 1, uma tensão superior a V_{OH} (***output high***). Por seu lado, as portas lógicas têm que interpretar qualquer tensão inferior a V_{IL} como sendo o nível lógico 0 e superior a V_{IH} como sendo o nível lógico 1 (Figura 6.21). Em resumo:

- V_{OH} – tensão de saída mínima que a porta fornece quando estiver ao nível alto.
- V_{OL} – tensão de saída máxima que a porta fornece quando estiver ao nível baixo.
- V_{IH} – tensão mínima que pode ser aplicada à entrada e reconhecida como nível alto.
- V_{IL} – tensão máxima que pode ser aplicada à entrada e reconhecida como nível baixo.

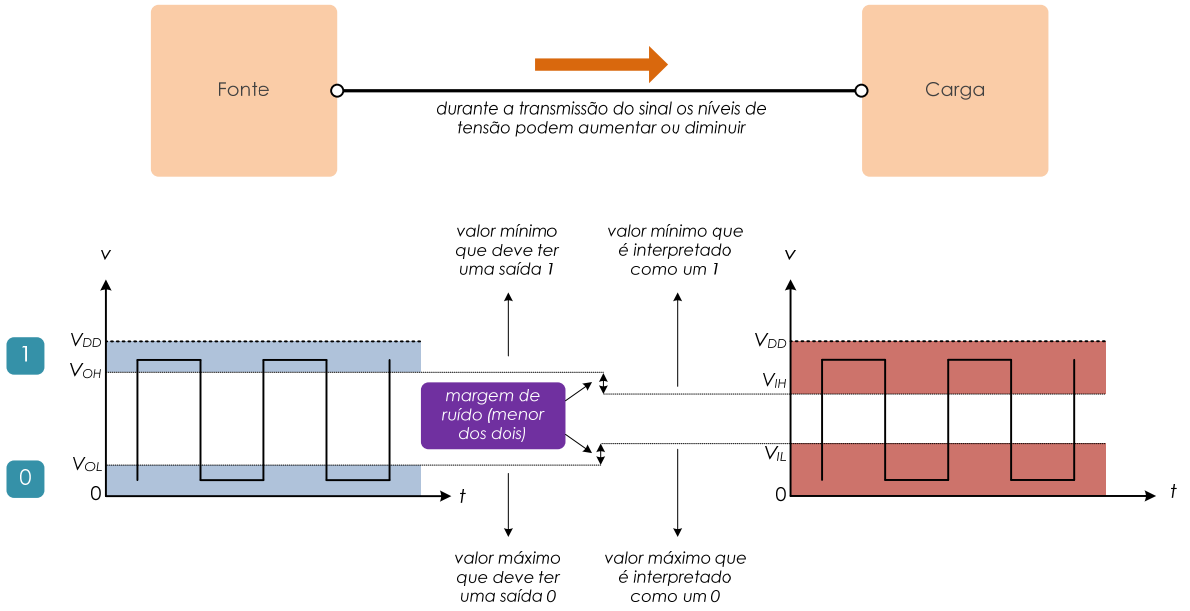


Figura 6.21 – Ilustração da definição de margem de ruído de um circuito digital.

A existência destas margens de ruído permite que os circuitos funcionem correctamente mesmo que a tensão produzida por uma porta lógica seja ligeiramente corrompida por ruído ou ondas reflectidas que provoquem oscilações no sinal.

A Figura 6.22 mostra os valores das margens de ruído para os circuitos lógicos TTL e CMOS operando a 5 V. Os circuitos CMOS têm margens de ruído maiores.

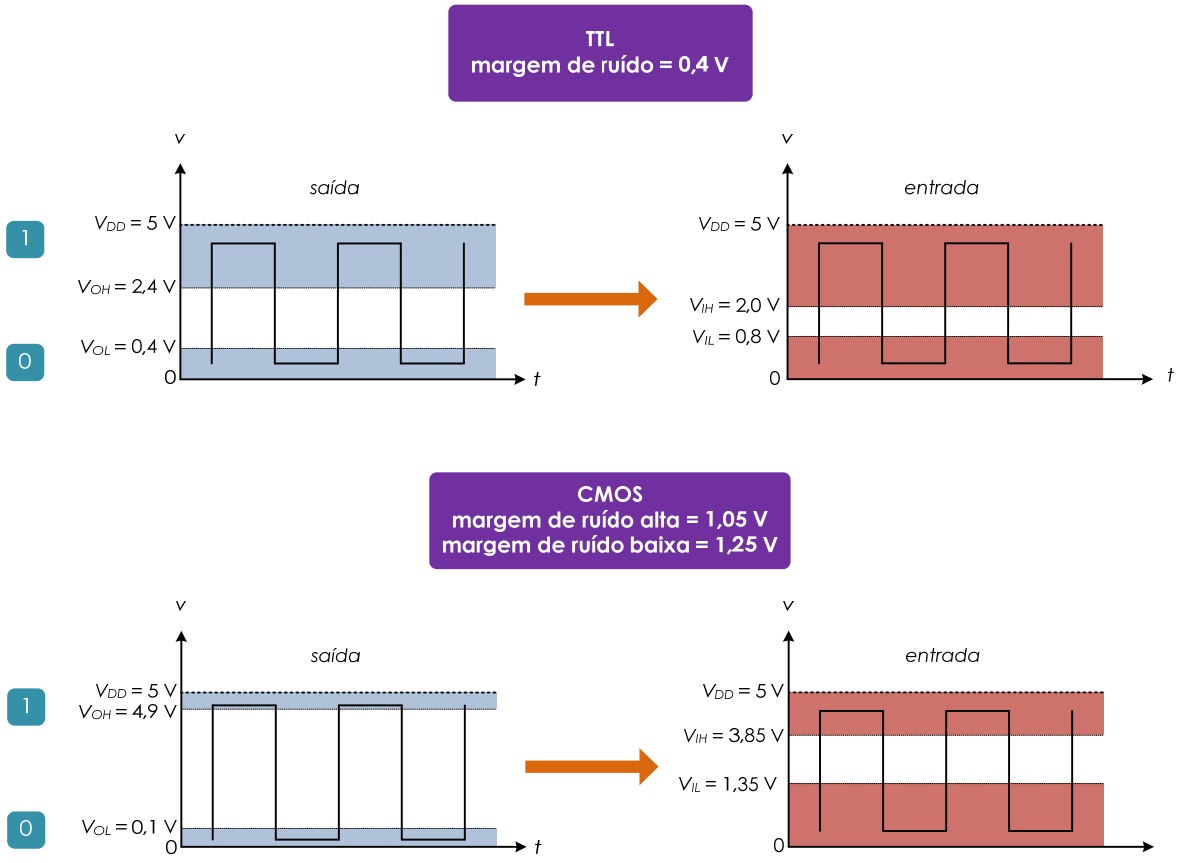


Figura 6.22 – Valores das margens de ruído dos circuitos digitais TTL e CMOS.

Ao usar-se uma terminação paralela ligada à massa a tensão de saída aquando do nível alto será inferior à que se teria sem terminação. No caso de ligação à alimentação positiva o nível baixo terá uma tensão superior. Isso faz com que as margens de ruído diminuam.

A terminação paralela tem também a desvantagem de desperdiçar energia na resistência inserida. No caso da resistência ligada à massa isso acontece quando o nível lógico é 1. No caso da resistência ligada à alimentação isso acontece com o nível lógico 0. A potência média dissipada é assim dada por

$$P_{med} = d\frac{V_{DD}^2}{R} \tag{46}$$

em que d é o valor do ciclo de trabalho, no caso em que a resistência da terminação está ligada à massa e é de

$$P_{med} = (1-d)\frac{V_{DD}^2}{R} \tag{47}$$

quando essa resistência está ligada a V_{DD}.

Considerando uma tensão de alimentação de 5 V, uma resistência de terminação de 100 Ω e um ciclo de trabalho de 50%, cada uma dessas soluções leva a uma potência dissipada de 125 mW. Este valor é muito mais elevado do que o que consome um transístor CMOS o que faz com que este tipo de terminações não seja utilizado, em geral, com circuitos CMOS.

A escolha da melhor solução em termos de potência dissipada depende portanto do valor médio do ciclo de trabalho dos sinais que existiram na linha do barramento. Ciclos de trabalho elevados favorecem a escolha da terminação cuja resistência está ligada a V_{DD} enquanto ciclos de trabalho reduzidos levam à escolha da terminação cuja resistência está ligada à massa.

Para evitar este tipo de escolha, baseado no ciclo de trabalho do sinal, pode-se usar um outro tipo de terminação consiste em combinar as duas soluções anteriores num tipo de terminação conhecida como sendo do tipo Thevenin. Ela é constituída por duas resistências – uma ligada à massa e outra à alimentação positiva (Figura 6.23). Naturalmente que este tipo tem a desvantagem de necessitar do dobro dos componentes.

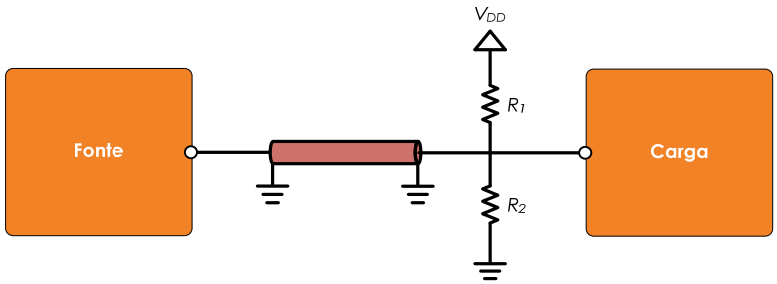


Figura 6.23 – Terminação Thevenin constituída por duas resistências ligadas à massa e à alimentação positiva.

A impedância vista pela terminação da linha de transmissão em direcção à carga é o paralelo de R_1 e R_2. Tem-se portanto que ter

$$\frac{R_1 R_2}{R_1 + R_2} = Z_0 . \tag{48}$$

Para além disso há que garantir que a corrente na saída da fonte não seja maior que ela consegue fornecer/absorver. Essa corrente depende do nível lógico da sua saída e é designada por I_{OHmax} ou I_{LHmax}:

- I_{OHmax} – Máxima corrente que pode ser fornecida pelo terminal de saída (***output***) quando esta se encontra no nível alto (***high***). Se a corrente absorvida for maior do que esta não é garantido que a tensão de saída seja maior do que V_{OHmin}.

- I_{OLmax} – Máxima corrente que pode ser absorvida pelo terminal de saída (*output*) quando esta se encontra no nível baixo (*low*). Se a corrente absorvida for maior do que este valor não é garantido que a tensão de saída seja menor do que V_{OLmax}.

Note-se que em algumas folhas de especificação de fabricantes não é indicado I_{OHmax} nem I_{OLmax} mas sim o valor recomendado da corrente de saída, definida de fora para dentro do circuito, quando o nível lógico é alto ou baixo. Essas correntes são designadas por I_{OH} e I_{OL} respectivamente. O valor de I_{OH} especificado é portanto negativo pois no nível alto o dispositivo fornece corrente à carga.

A Figura 6.24 ilustra o caso em que o nível lógico é baixo e em que a carga não consome nem fornece nenhuma corrente no seu terminal de entrada.

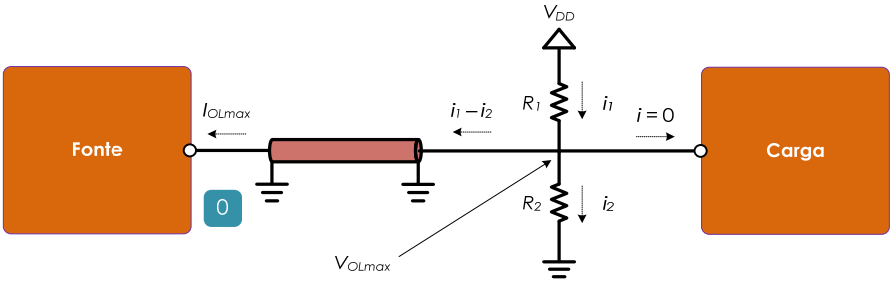


Figura 6.24 – Indicação das tensões e correntes numa terminação Thevenin quando a saída esta ao nível baixo.

A corrente que sai da terminação (divisor de tensão R_1/R_2), que vale $i_1 - i_2$, não pode ser maior do que a corrente que a fonte é capaz de absorver para garantir que a sua saída não é maior do que V_{OLmax},

$$i_1 - i_2 < I_{OL\max} \,. \tag{49}$$

Nessas condições (49) fica

$$\frac{V_{DD} - V_{OL\max}}{R_1} - \frac{V_{OL\max}}{R_2} < I_{OL\max} \,. \tag{50}$$

Por seu lado, quando a saída encontra-se no nível lógico alto, a tensão de saída não deve ser menor do que V_{OHmin} e a corrente máxima que a fonte é capaz de fornecer, para que isso aconteça, é I_{OHmax} (Figura 6.25).

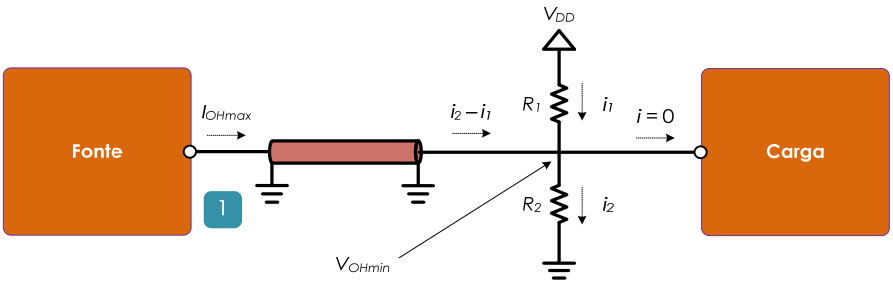


Figura 6.25 – Indicação das tensões e correntes numa terminação Thevenin quando a saída esta ao nível baixo.

É portanto necessário garantir que

$$i_2 - i_1 < I_{OH\max}\,, \tag{51}$$

o que leva a

$$\frac{V_{OH\min}}{R_2} - \frac{V_{DD} - V_{OH\min}}{R_1} < I_{OH\max}\,. \tag{52}$$

Escrevendo as equações (48), (50) e (52) com $1/R_1$ (G_1)em função de $1/R_2$ (G_2) obtém-se

$$\begin{aligned} G_1 &= \frac{1}{Z_0} - G_2 \\ G_1 &< \frac{G_2 V_{OL\max} + I_{OL\max}}{V_{DD} - V_{OL\max}}\,. \\ G_1 &> \frac{G_2 V_{OH\min} - I_{OH\max}}{V_{DD} - V_{OH\min}} \end{aligned} \tag{53}$$

Estas equações estão representadas na Figura 6.26. Os valores permitidos para G_1 e G_2 são os que se encontram sobre o segmento de recta que liga os pontos A e B.

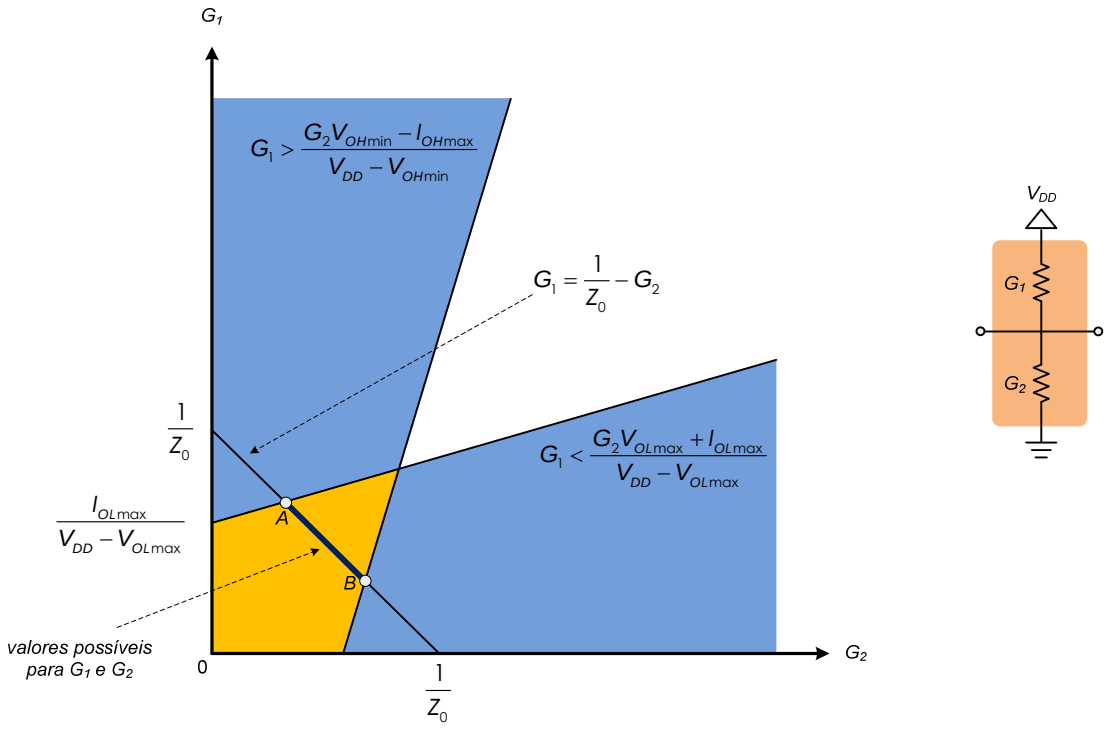


Figura 6.26 – Determinação gráfica dos valores possíveis para as resistências da terminação Thevenin.

Tendo em conta o que se disse anteriormente sobre o facto de que em circuitos CMOS é comum a capacidade de um circuito de fornecer corrente é menor do que a sua capacidade de absorver corrente, é comum escolher valores de R_1 e R_2 que levam a que a tensão na terminação (V_T) seja a

tensão mínima de saída no nível alto, ou seja, v_{OHmin}. Isso corresponde ao ponto B na Figura 6.26 e a valores de resistência da terminação dados por

$$\begin{aligned} R_1 &= Z_0 \frac{2V_{OH\min} - V_{DD}}{V_{OH\min} + I_{OH\max} Z_0} \\ R_2 &= \frac{R_1 Z_0}{R_1 - Z_0} \end{aligned} \quad . \tag{54}$$

Neste tipo de terminação a resistência R_1 ajuda a "puxar" a tensão para cima nos níveis altos e a resistência R_2 ajuda a"puxar" a tensão para baixo nos níveis baixo. Isso leva a uma melhoria das margens de ruído. Da mesma forma essas resistências ajudam a fornecer e absorver a corrente da carga aumentando o *fan-out*.

A principal desvantagem deste tipo de terminação tem a ver com a corrente DC que consome independentemente do nível lógico. Outra desvantagem tem a ver com a necessidade de ligação tanto à massa como à alimentação o que implica mais ligações no circuito.

Um outro tipo de terminação consiste na colocação de uma resistência em série com a fonte como ilustrado na Figura 6.27.

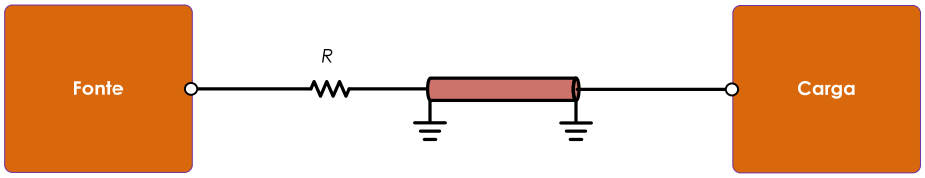


Figura 6.27 – Terminação série.

O valor da resistência a colocar tem que ser

$$R = Z_0 - R_D \, , \tag{55}$$

em que R_D é a impedância de saída da fonte, de modo a que a resistência vista pelas ondas reflectidas seja igual à impedância característica da linha.

O valor da onda incidente é metade do valor da tensão de saída da fonte. Como a impedância da carga é em geral alta, surge uma onda reflectida com uma amplitude semelhante à da onda incidente.

Para diminuir o consumo da terminação paralela, causada pela corrente DC que percorre a resistência, pode-se inserir, em série com esta, um condensador, como ilustra a Figura 6.28, de modo a eliminar essa corrente. Este tipo de terminação é conhecido por *Terminação AC*.

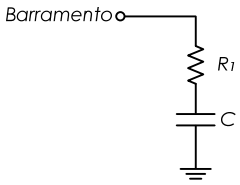


Figura 6.28 – Terminação AC.

O valor da resistência deve ser igual ao valor da impedância característica da linha de modo a eliminar reflexões.

Em alguns casos é importante também garantir que a tensão na linha não tenha um valor negativo, causado por reflexões, pois isso pode levar a um funcionamento incorrecto dos circuitos que a ela estão ligados. Uma forma de garantir isso é com o circuito da Figura 6.29 usado, por exemplo, nos barramentos Multibus e Unibus.

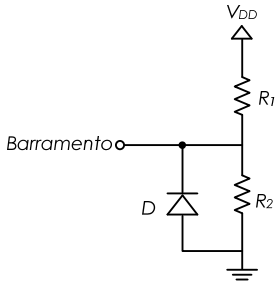


Figura 6.29 – Terminação Thevenin com díodo.

O paralelo de R_1 e R_2 deve ser igual à impedância característica da linha (Z_0).

Neste caso a tensão na linha nunca desce abaixo do valor da tensão directa do diodo que no caso de um díodo normal é aproximadamente 0,7 V e num díodo Schottky está entre 0,15 V e 0,45 V.

Um outro tipo de terminação com o mesmo objectivo é o usado no barramento Eurobus e ilustrado na Figura 6.30. Neste caso a tensão na linha nunca é negativa.

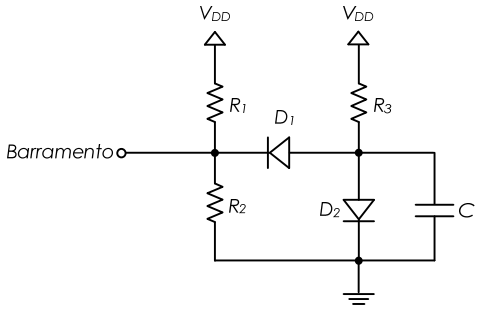


Figura 6.30 – Terminação Thevenin com dois díodos e condensador.

A resistência R_3 polariza o díodo D_2 directamente a tensão no seu ânodo (e no ânodo de D_1) é pois aproximadamente 0,7 V. Quando a tensão no barramento desce abaixo de 0 díodo D_1 fica também

polarizado directamente o que faz com que a tensão no seu cátodo nunca seja inferior a 0 (0,7 inferior à tensão no seu ânodo que tem o valor de 0,7 V devido ao díodo D_2).

A terminação da Figura 6.31 utiliza um diodo de Zener (DZ_1), polarizado inversamente por R_1, de modo a impor uma tensão na linha do barramento (V_T).

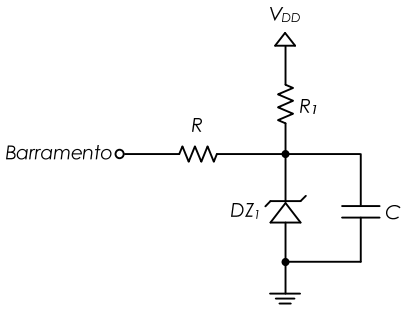


Figura 6.31 – Terminação usando um díodo de Zener.

No nível lógico alto, e assumindo que a tensão na linha vale V_{DD} (pior caso), a potência dissipada na resistência da terminação (R) é

$$P_{1'} = \frac{\left(V_{DD} - V_T\right)^2}{R}. \tag{56}$$

em que V_T é a tensão aos terminais do díodo de Zener. No caso do nível lógico baixo, e assumindo que a tensão na linha é 0 (pior caso) a potência dissipada é

$$P_{0'} = \frac{\left(0 - V_T\right)^2}{R}. \tag{57}$$

A potência total, considerando um ciclo de trabalho d, é portanto

$$P = d \cdot P_{1'} + \left(1-d\right) \cdot P_{0'} = d\frac{\left(V_{DD} - V_T\right)^2}{R} + \left(1-d\right)\frac{V_T^2}{R} \tag{58}$$

Para obter o valor de V_T que minimiza a potência consumida há que determinar a derivada de P é ordem a V_T, e igualar a 0 (extremo da função). Tem-se assim

$$\frac{dP}{dV_T} = \frac{1}{R}\left[-2d\left(V_{DD} - V_T\right) + 2\left(1-d\right)V_T\right]. \tag{59}$$

Igualando a 0 obtém-se

$$\frac{dP}{dV_T} = 0 \Leftrightarrow 2\left(1-d\right)V_T = 2d\left(V_{DD} - V_T\right) \Leftrightarrow V_T = dV_{DD}. \tag{60}$$

A solução é portanto ter um Zener com uma tensão de $d \cdot V_{CC}$. Note-se que este valor corresponde ao valor médio da tensão na linha o que é válido mesmo que a tensão na linha não varie entre 0 e V_{CC} como é o caso da derivação efectuada.

O equivalente de Thevenin da terminação da Figura 6.31 é resistência *R* ligada a uma fonte de tensão com o valor V_T. Relembre-se que para obter o valor da resistência do esquema equivalente de Thevenin o díodo de Zener comporta-se como uma fonte e tensão (pois está a impor uma tensão aos seus terminais quando está polarizado inversamente) e portanto deve ser curto-circuitado.

Conclui-se assim que nesta terminação O valor da resistência *R* deve ser igual a Z_0 e que o valor da tensão do díodo de Zener deve ser igual ao valor média da tensão na linha do barramento.

A terminação da Figura 6.32 é uma terminação activa que usa um regulador para impor uma tensão DC na linha o que permite diminuir o consumo de energia tal como acontece com o uso do díodo de Zener. O valor da tensão V_T usado é, no barramento S-100, o valor da tensão no nível alto V_{OH}. A resistência tem de ter um valor igual ao da impedância característica da linha.

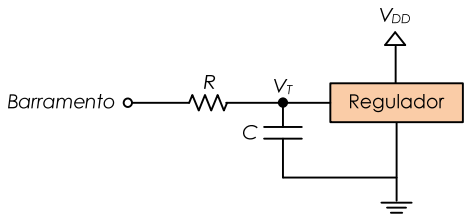


Figura 6.32 – Terminação activa com regulador.

Outra solução para uma terminação activa é o uso de um amplificador operacional como ilustrado na Figura 6.33.

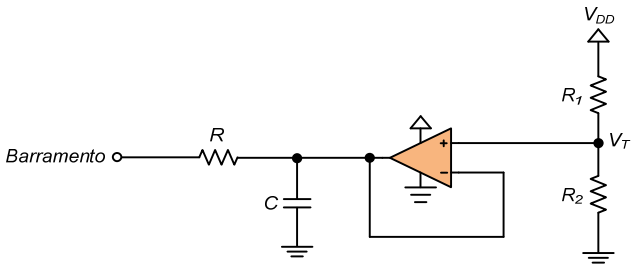


Figura 6.33 – Terminação usando um amplificador operacional.

O amplificador diferencial é usado em montagem seguidora de forma a impor na linha a tensão (V_T) determinada pelo divisor resistivo constituído por R_1 e R_2. A resistência *R* deve ter um valor igual a Z_0 para que a linha esteja adaptada.

Os dimensionamentos efectuados para as terminações são feitos de modo a que as linhas do barramento estejam adaptadas quando não existe nenhuma placa conectada a este. A introdução de uma placa no barramento faz aumentar a capacidade deste. Esse aumento vai alterar a sua

impedância característica que deixará de ser igual à impedância da terminação e portanto levará a que a linha fique desadaptada. Essa desadaptação traduz-se na existência de uma onda reflectida que ao ser somada com a onda incidente vi provocar uma variação da tensão total ao longo da linha. Essa variação tem que ser menor do que a margem de ruído admitida para que todos os dispositivos ligados ao barramento funcionem correctamente.

A variação da tensão na linha pode ser expressa pelo factor de onda estacionário

$$s = \frac{V_{max}}{V_{min}} . \tag{61}$$

No caso do nível lógico alto a onda incidente vale, no pior caso V_{OH}. Tem-se portanto um valor máximo para o factor de onda estacionária de

$$s_{max}^{'1'} \leq \frac{VO_H + NM_H}{VO_H - NM_H} . \tag{62}$$

No caso do nível lógico baixo a onda incidente vale, no pior caso V_{OL} e portanto tem-se.

$$s_{max}^{'0'} \leq \frac{VOL + NM_L}{VO_L - NM_L} . \tag{63}$$

Normalmente os valores das margens de ruído NM_H e NM_L são semelhantes enquanto o valor de V_{OL} é muito menor do que V_{OH}. Isso faz com que o valor limite do factor de onda estacionária dado por (63) seja maior que o valor limite dado por (62). Seja como for deve-se escolher como limite o menor dos dois valores.

$$s_{max} = \min\left(s_{max}^{'0'}, s_{max}^{'1'}\right) . \tag{64}$$

O factor de onda estacionária, por sua vez, depende da impedância característica da linha (Z_0) e da impedância da terminação (Z_L):

$$s = \begin{cases} \frac{Z_L}{Z_0} & , \ Z_L \geq Z_0 \\ \frac{Z_0}{Z_L} & , \ Z_L < Z_0 \end{cases} .$$

Como a introdução de capacidade na linha vai fazer diminuir o valor da impedância característica a expressão que interessa é a primeira:

$$s = \frac{Z_L}{Z_0} \quad , \ Z_L \geq Z_0 .$$

Combinando o valor de *s* máximo, o valor da impedância da terminação (Z_L) e a expressão para a impedância característica da linha com a introdução de *n* placas com uma capacidade C_x, obtém-se

$$\frac{Z_L}{\sqrt{\frac{L}{C + n \times C_x}}} \leq s_{max} \,. \tag{65}$$

Resolvendo-se em ordem ao valor da capacidade das placas obtém-se

$$n \cdot C_x \leq \frac{L}{\left(\frac{Z_L}{s_{max}}\right)^2} - C \,. \tag{66}$$

Note-se que os valores de L e C que aparecem em (66) são os valores da indutância e capacidade característicos da linha multiplicados pelo seu comprimento.

6.4.3 Sinalização Diferencial

Um sinal eléctrico de tensão necessita de dois condutores pois a tensão não é mais do que a diferença de potencial entre dois pontos. Um sinal de corrente necessita na mesma de dois condutores pois a corrente eléctrica tem que percorrer um circuito fechado. É pois necessário um condutor para levar a corrente de uma fonte para uma carga e de outro condutor para a trazer de volta à fonte.

Tipicamente os circuitos electrónicos utilizam um condutor comum como referência para todos os sinais nesse circuito que necessitam assim de um único condutor para interligação entre os diversos componentes. A vantagem naturalmente é a simplicidade de implementação e uma redução de custo devido ao menor comprimento de cabo utilizado.

Esse tipo de sinalização de modo comum (***single-ended***), usado por exemplo no protocolo RS-232 (porta série dos computadores) não permite a operação com elevadas frequências pois o efeito da indutância e capacidade distribuídas pelos condutores é semelhante a um filtro passa-baixo. Por outro lado o uso de tensões elevadas em cabos compridos requer uma potência de transmissão elevada. Isso pode reduzir-se baixando os valores de tensão utilizados mas a determinada altura o ruído presente acaba por corromper os sinais mesmo sendo digitais.

Quando se pretende operar a alta frequência de modo a aumentar a quantidade de informação transmitida a solução adoptada é usar sinalização diferencial (***differential signaling***), isto é, usar dois condutores para cada sinal. Este método é usado, por exemplo, nos barramentos PCI-Express para ligação de placas de expansão a computadores de secretária, no protocolo USB para ligação a periféricos e no protocolo Serial ATA para ligação a discos rígidos.

No protocolo USB existe só um par de condutores para os dados (e mais dois para a alimentação). Os vários bits têm portanto de ser transmitidos em série um a seguir ao outro. Em outros protocolos que

necessitam de uma maior taxa de transferência de dados, como o PCI-Express ou o HyperTransport, estes são transmitidos em paralelo usando-se vários pares de condutores, um para cada bit.

Um dos sistemas de sinalização diferencial mais comum é o LVDS (*low-voltage differential signaling*). O transmissor injecta uma corrente de 3,5 mA num dos condutores de forma a transmitir um '1' ou um '0'. Do lado do receptor existe uma resistência com um valor igual à impedância característica da linha de transmissão (tipicamente 100 Ω) de modo a provocar uma queda de tensão da ordem do ±350 mV (Figura 6.34). O receptor determina a polaridade do sinal de forma a interpretá-lo como sendo o símbolo '0' ou o símbolo '1'. A tensão de modo comum mantida nos dois condutores é aproximadamente 1,25 V.

Figura 6.34 – Diagrama de uma ligação diferencial do tipo LVDS.

REFERÊNCIAS

[1] "Fundamental Principles of Switching Circuits and Systems", AT&T Bell Telephone Laboratories, 1961.

[2] Lance Zheng, "Electronics: Memory for Micros", Appliance Design Webpage, 2008. Extraído em 27 de Julho de 2009 de http://www.appliancedesign.com/Articles/Electronics.

[3] Akhilesh Tyagi, "A Reduced-Area Scheme for Carry-Select Adders", IEEE Transactions on Computers, vol. 42, nº. 10, Outubro de 1993, pp. 1163-1170.

[4] Mike Hally, *Electronic Brains: Stories from the Dawn of the Computer Age*, Joseph Henry Press, 2005.

[5] Alice R. Burks, Arthur W. Burks, "The first Electronic Computer – The Atanasoff Story", The University of Michigan Press, 1988.

[6] Behrooz Parhami, "Computer Arithmetic – Algorithms and Hardware Design", Oxford University Press, 2000.

[7] Karthik Ethirajan, John Nemec, "Termination Techniques for High-speed Buses", Electronic Design, Strategy, News (EDN), Fevereiro de 1998.

[8] http://www.eng.utah.edu/~cs5830/Slides/chap5x6.pdf

[9] Milos Ercegovac e Tomas Lang, "On the fly conversion of redundant into conventional representations", Agosto de 1985.

[10] David Tawei Wang, "Modern DRAM Memory Systems: Performance Analysis and Scheduling Algorithm", Tese de Doutoramento, University of Maryland, 2005.