1 Introdução

Um sistema pode ser definido como sendo um conjunto de elementos que são interligados de alguma maneira para compor um todo e assim realizar funcionalidade específica. Por exemplo, um aparelho de som hi-fi é composto de vários componentes, tais como compartimento para discos e fitas, amplificador e auto-falantes. Todos são interconectados por cabos elétricos.

Um sistema também possui uma função bem definida, a qual pode ser identificada a partir das funcionalidades de seus componentes. Por exemplo, a função do aparelho de som hi-fi é transformar a informação armazenada em discos e/ou fitas em som audível, o que é algo que nenhum dos componentes do sistema pode realizar por si só.

Neste sentido, pode-se identificar dois aspectos fundamentais em qualquer sistema: sua estrutura e seu comportamento. A estrutura reflete os componentes e como eles estão interconectados e o comportamento reflete a funcionalidade do sistema.

Sistemas em que o número de componentes é alto e/ou as inter-relações entre eles não são muito claras ou difíceis de serem estabelecidas e entendidas são ditos complexos. No projeto de tais sistemas complexos, a identificação de alguma ordem ou regularidade é de extrema importância, o que normalmente requer uma abordagem estruturada e sistemática.

Dependendo de quais aspectos se está tentando identificar, deve-se usar um tipo de representação e um nível de abstração adequados. Os três tipos mais comuns de representação são o **comportamental**, o **estrutural** e o **físico**.

Uma **representação comportamental** captura o sistema como uma caixa preta e se concentra na especificação do comportamento como uma função dos valores de entrada e o tempo (ver figura 1.1). Em outras palavras, uma representação comportamental descreve a funcionalidade mas não a implementação de um dado sistema, definindo as respostas da caixa preta para qualquer combinação dos valores de entrada mas sem descrever como projetar ou construir o sistema usando dados componentes.

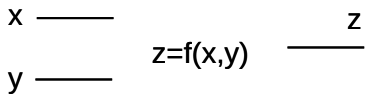

Figura 1.1 - Sistema como uma caixa preta.

Uma **representação estrutural** define a caixa preta como um conjunto de componentes e suas interconexões. Diferente da representação comportamental, a representação estrutural especifica a implementação do sistema sem qualquer referência à sua funcionalidade. Isto é ilustrado na figura 1.2.

Obviamente, muitas vezes a funcionalidade pode ser derivada a partir dos componentes interconectados. No entanto, derivar a funcionalidade de um sistema desta maneira é muito difícil, principalmente se o sistema possui um grande número de componentes.

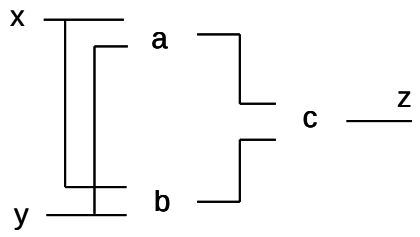

Figura 1.2 - Sistema como um conjunto de componentes interconectados.

Uma **representação física** especifica as características físicas da caixa preta, indicando as dimensões e as posições de cada componente e conexão existentes na descrição estrutural do sistema. Enquanto a representação estrutural fornece a conectividade do sistema, somente a representação física é que descreve precisamente as relações espaciais dos vários componentes. Ou seja, a representação física é usada para descrever o sistema depois que ele foi fabricado, especificando seu peso, tamanho, consumo de potência e posição de cada pino de entrada ou saída.

Uma maneira natural e conveniente para representar a estrutura de um sistema é o uso de diagramas de bloco, onde componentes são representados por retângulos, chamados blocos, e conexões que são denotadas por linhas interligando os blocos. A figura 1.3 mostra um diagrama do aparelho de som hi-fi.

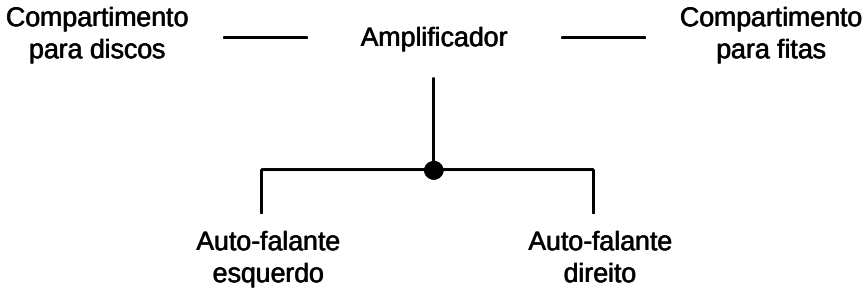

Figura 1.3 - Diagrama de blocos para o aparelho de som hi-fi.

O processo de projeto de sistemas, principalmente produtos eletrônicos em geral e sistemas digitais em particular, consiste sempre de pelo menos três fases, cada uma centrada em uma das representações de projeto:

1. derivar uma representação comportamental da funcionalidade do produto
2. converter esta representação para uma representação estrutural contendo componentes de uma dada biblioteca de componentes
3. produzir uma representação física que especifica como montar e fabricar o produto

Qualquer projeto pode ser realizado seguindo estes passos usando diferentes níveis de abstração. Em um nível de abstração apenas determinados detalhes são representados. Normalmente, os detalhes capturados em uma dada fase do projeto dependem da complexidade do sistema. Por exemplo, é praticamente impossível projetar um microprocessador inteiro usando apenas portas lógicas básicas. Normalmente, inicia-se o projeto pelos blocos básicos no nível lógico. A seguir, estes blocos são interconectados para compor um sistema mais complexo, como por exemplo um microprocessador.

Na próxima seção, serão abordados outros formalismos para a representação da estrutura e do comportamento de sistemas, além dos níveis de abstração mais comumente utilizados no projeto de sistemas digitais.

1.1 Sistemas Digitais

Este texto tem por objetivo introduzir o estudo de sistemas digitais, dando especial atenção aos componentes básicos destes sistemas. Um sistema digital pode ser definido como um conjunto de componentes interconectados que processam informações em forma digital ou discreta. Na maioria dos sistemas digitais, os componentes básicos utilizados são dispositivos eletrônicos chamados circuitos integrados (CIs). As ligações entre estes componentes eletrônicos são conexões físicas através das quais a informação digital pode ser transmitida.

Sistemas digitais modernos abrangem uma vasta gama de graus de complexidade. Os componentes disponíveis para a construção de sistemas digitais vão desde chaves do tipo liga-desliga até computadores completos. O número de componentes em um sistema digital pode variar de um, dois ou de milhares de componentes. Obviamente, quanto mais componentes são necessários à implementação de um sistema digital, mais complexo ele é e, conseqüentemente, mais difícil de entender seu funcionamento e de projetá-lo. Daí a importância do uso de níveis de abstração no processo de projeto de sistemas digitais.

Um nível de abstração, ou de granularidade, é caracterizado pelo tipo de objetos utilizados na representação. Em geral, pode-se identificar quatro diferentes tipos de objetos em um produto eletrônico: transistores, portas, registradores e processadores. Os correspondentes níveis de abstração são sumarizados na tabela 1.1.

Tabela 1.1 – Níveis de abstração.

Nível	Comportamento	Estrutura	Físico
Transistor	Equações diferenciais, diagramas corrente-voltagem	Transistores, resistores, capacitores	Células analógicas e digitais
Portas	Equações Booleanas, máquinas de estado finitas (FSM)	Portas lógicas, Flip-flops	Módulos, unidades
Registrador	Algoritmos, *flowcharts*, conjunto de instruções, generalizações de FSMs	Somadores, comparadores, contadores, registradores	Microcircuitos
Processador	Especificação executável, programas	Processadores, controladores, ASICs	Placas de circuito impresso, módulos multicircuitos

Como mostrado na tabela 1.1, os principais componentes no **nível de transistores** são transistores, resistores e capacitores, que são combinados para formar circuitos analógicos e digitais que realizam uma dada funcionalidade. Esta funcionalidade é usualmente descrita por um conjunto de equações diferenciais ou por algum tipo de relacionamento entre corrente e tensão. A representação física destes circuitos, denominados células, podem consistir de componentes do nível de transistores e as conexões que os interconectam.

Os principais componentes do **nível de portas** são portas lógicas e flip-flops. Portas lógicas são circuitos especiais que implementam operações Booleanas, tais como E e OU. Um flip-flop é um elemento básico de memória que é capaz de armazenar um bit de informação, podendo assumir o valor 0 (falso) ou 1 (verdadeiro). Portas e flip-flops são células digitais que podem ser grupadas para formar módulos ou unidades aritméticas e de memória. Os módulos são utilizados como componentes básicos no nível de registradores. O comportamento de cada módulo pode ser descrito com o uso de equações Booleanas e diagramas de máquinas de estados finitas (Finite State Machines - FSMs).

No **nível de registradores**, os principais componentes são unidades aritméticas e de memória, tais como, somadores, comparadores, multiplicadores, contadores, registradores, bancos de registradores, filas, etc. Cada um destes componentes é um módulo com dimensões fixas, um atraso de propagação e um conjunto de posições (fixas) para as entradas e saídas do módulo. Estes componentes do nível de registradores podem ser montados e interconectados em microcircuitos, que podem ser usados como componentes básicos no nível de abstração acima. Usualmente, estes microcircuitos são descritos por fluxogramas, conjuntos de instruções, diagramas de FSMs ou tabelas de estados.

O mais alto nível de abstração apresentado na tabela é o **nível de processador**, onde os componentes básicos são processadores, memórias, controladores e interfaces, além de circuitos de aplicação específica (*Application Specific Integrated Circuits* - ASICs). Geralmente, um ou mais destes componentes são montados em uma placa de circuito impresso e conectados com fios que são impressos na placa. O comportamento de sistemas compostos destes componentes do nível de processadores é usualmente descrito utilizando-se uma linguagem natural, uma especificação executável em alguma linguagem de descrição de hardware (*Hardware Description Language* - HDL), ou um algoritmo ou programa escrito em uma linguagem de programação convencional.

A representação estrutural mais comum para sistemas digitais é o diagrama de blocos. Dependendo do nível de abstração, cada bloco pode representar objetos correspondentes ao nível, por exemplo, portas lógicas no nível de abstração de portas; registradores, somadores, etc. no nível de registradores; e controladores e ASICs no nível de processador. As interconexões representam as conexões entre os objetos.

A figura 1.4 mostra o diagrama de blocos para a **unidade lógica e aritmética** (ULA) e os **registradores de entrada e saída**, os quais constituem partes essenciais de qualquer sistema computacional. No diagrama de blocos, não há detalhamento dos componentes, mas apenas uma idéia genérica da estrutura como um todo. Cada bloco pode ser mostrado com mais detalhes, em algum outro diagrama de blocos mais detalhado, ou num diagrama esquemático. Isto revela a natureza hierárquica existente por detrás de qualquer projeto de sistema.

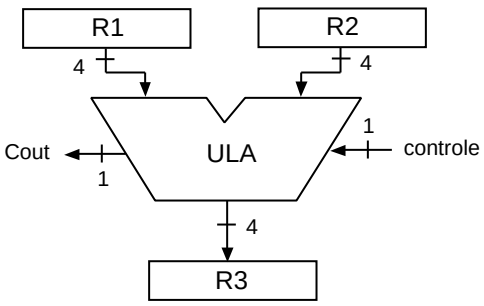

Figura 1.4 - Diagrama de blocos genéricos de uma ULA e seus registradores.

A figura 1.5 mostra o diagrama de blocos mais detalhado para a unidade lógico-aritmética (ULA) da figura 1.4. Já a figura 1.6 detalha o somador completo, que é o tipo de bloco usado para compor a ULA.

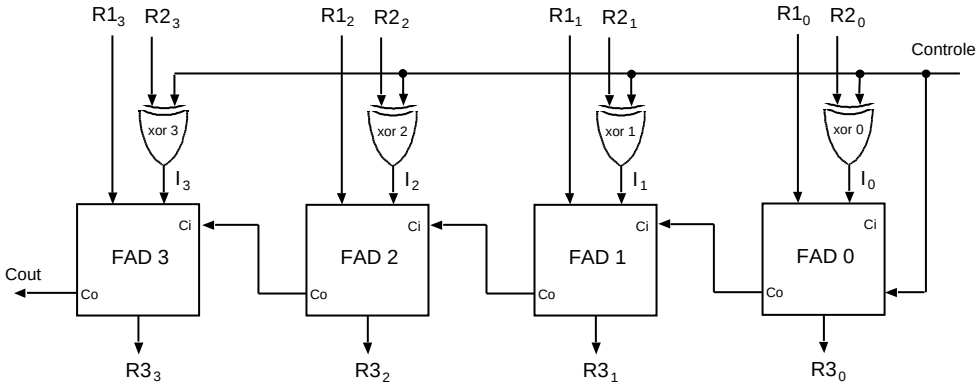

Figura 1.5 - Diagrama de blocos mais detalhado de uma ULA.

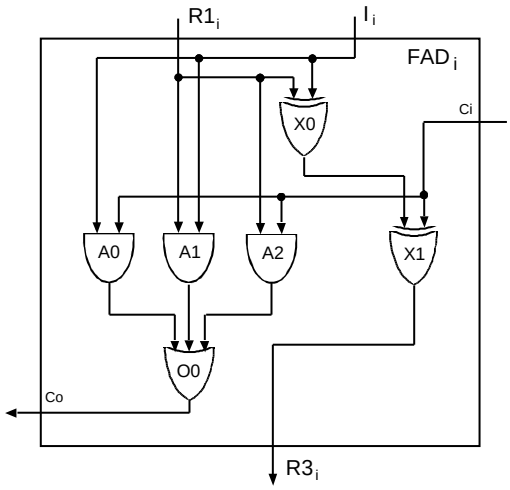

Figura 1.6 - Esquemático de um somador completo (*full-adder* - FAD).

Para representar o comportamento de sistemas digitais no nível de portas lógicas usam-se tabelas verdades e equações Booleanas. No nível de registradores, usa-se algum tipo de linguagem para descrever as transferências de dados entre os registradores. Para possibilitar a simulação do sistema no nível de processador são geralmente utilizadas linguagens procedurais para descrever os algoritmos na forma de programas executáveis.

Por exemplo, para a estrutura da ULA dada na figura 1.4, a descrição algorítmica poderia ser:

```
se (controle=0)
   R3_ R1+R2;
senão
   R3_ R1-R2;
```

Esta descrição indica que o valor de `controle` determina se a ULA realizará uma soma ou uma subtração.

1.2 Tipos de Dados e Representação de Dados

Nesta seção, iremos estudar os tipos de dados mais comumente encontrados nos sistemas digitais, mostrando como eles podem ser codificados em formato binário. Os dados encontrados nos sistemas digitais (note-se que os computadores atuais são sistemas digitais) podem ser classificados em uma das seguintes categorias:

- Números usados em cálculos aritméticos;
- Letras do alfabeto, usadas no processamento de dados;
- Símbolos discretos, usados para diversos propósitos

Todos os dados são representados em formato binário porque o uso deste formato facilita o projeto de circuitos eletrônicos que exibem duas condições possíveis, as quais são convenientemente interpretadas como os valores 0 e 1 de um dígito binário (bit). Tais circuitos eletrônicos são projetados para realizar um repertório de operações necessárias que são disponibilizadas nos computadores.

1.2.1 Notação Posicional

Todos os sistemas numéricos utilizados pelo ser humano são posicionais. Em um sistema posicional, cada dígito possui um peso associado. Assim, o valor de um dado número corresponde a uma soma ponderada de seus dígitos, como por exemplo:

$$1999 = 1 \times 1000 + 9 \times 100 + 9 \times 10 + 9 \times 1$$

Note que, no número anterior, o peso de cada posição é 10^i, onde i corresponde à posição do dígito, contada a partir da direita, e sendo i=0. para o dígito mais à direita.

Em geral, qualquer número decimal D , no formato $d_1\ d_0\ d_{-1}\ d_{-2}$ tem como valor:

$$D = d_1 \times 10^1 + d_0 \times 10^0 + d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2}$$

onde 10 é chamado base.

Num sistema posicional genérico, a base pode ser qualquer valor inteiro *r*, e um dígito numa posição i assume peso r^i . Logo, podemos escrever o formato genérico de um número em tal sistema como sendo

$$d_{m-1}\ d_{m-2} \ldots d_1\ d_0\ .\ d_{-1}\ d_{-2} \ldots d_{-n}$$

onde há m dígitos à esquerda do ponto e n dígitos à direita do ponto. Note que, se não houver ponto, assume-se que este está à direita do dígito mais à direita. O valor deste número é o somatório dos produtos de cada dígito pela correspondente potência da base:

$$D = \sum_{i=-n}^{m-1} d_i \times r^i$$

Para um número qualquer, o dígito mais à direita é comumente referenciado como **dígito menos significativo** (*least-significative bit* - LSB, em inglês), ao passo que o dígito mais à esquerda é denominado **dígito mais significativo** (*most-significative bit* – MSB, em inglês).

Desde que sistemas digitais usam dígitos binários, estaremos particularmente interessados no sistema binário. Neste sistema, a base utilizada é **2**, de modo que um número em binário assume a forma:

$$b_{m-1}\, b_{m-2} \ldots b_1\, b_0 \,.\, b_{-1}\, b_{-2} \ldots b_{-n}$$

e seu valor pode ser encontrado por

$$B = \sum_{i=-n}^{m-1} b_i \times 2^i$$

Similarmente ao sistema decimal, o ponto no sistema binário é denominado **ponto binário**. Normalmente, quando se trabalha com sistemas de base não-decimal, indica-se a base subscrevendo-se o valor da base à direita do número. Exemplos:

$$10101_2 = 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 21_{10}$$

$$.1111_2 = 1 \times 0.5 + 1 \times 0.25 + 1 \times 0.125 + 1 \times 0.0625 = 0.9375_{10}$$

1.2.2 Números Octais e Hexadecimais

Além do sistema decimal e do sistema binário, dois outros sistemas são de grande importância por proverem representações convenientemente compactas de números grandes. Trata-se dos sistemas **octal** (base 8) e **hexadecimal** (base 16).

No sistema octal, cada dígito representa um valor entre 0 e 7. Já no sistema hexadecimal, cada dígito representa um valor entre 0 e 15. Para representar os valores maiores do que 9 usando apenas um dígito, utilizam-se letras. Assim, o valor 10 é representado por A, o 11, por B e assim por diante, até 15 (que é representado por F). A tabela 1.2 mostra os inteiros binários de 0 a 10001, juntamente com os equivalentes octal, decimal e hexadecimal. Note que cada dígito octal pode ser representado por 3 dígitos binários, enquanto que um dígito hexadecimal pode ser representado por 4 dígitos binários.

Tabela 1.2 - Representação de inteiros em binário, octal, decimal e hexadecimal.

Binário	Octal	Decimal	Hexa
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6

111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11

A tabela 1.3 mostra os inteiros binários de 0 a 10001 e seus equivalentes octal, decimal e hexadecimal codificados em binário.

Tabela 1.3 - Representação de inteiros em binário, octal, decimal e hexadecimal.

Binário	**Octal codificado em binário**	**Decimal codificado em binário**	**Hexadecimal codificado em binário**
0	000	0000	0000
1	001	0001	0001
10	010	0010	0010
11	011	0011	0011
100	100	0100	0100
101	101	0101	0101
110	110	0110	0110
111	111	0111	0111
1000	001 000	1000	1000
1001	001 001	1001	1001
1010	001 010	0001 0000	1010
1011	001 011	0001 0001	1011
1100	001 100	0001 0010	1100
1101	001 101	0001 0011	1101
1110	001 110	0001 0100	1110
1111	001 111	0001 0101	1111
10000	010 000	0001 0110	0001 0000
10001	010 001	0001 0111	0001 0001

Para se converter um número binário em octal separam-se os dígitos em grupos de 3, a partir do ponto binário para a esquerda. Após, substitui-se cada grupo pelo dígito octal correspondente. Exemplo:

$$1010011100_2 = 001\ 010\ 011\ 100 = 1234_8$$

E para converter-se um número em binário para hexadecimal, o procedimento é análogo, exceto que os grupos deverão ser de 4 dígitos.

Exemplo 1.1: converter-o número binário que segue para hexadecimal

$$1010011100_2 = 0010\ 1001\ 1100 = 29C_{16}$$

Note que nestes exemplos foram adicionados zeros à esquerda, de modo que todos os grupos tivessem 3 dígitos, no caso da conversão direta binário-octal, e 4 dígitos no caso da conversão direta binário-hexadecimal.

A conversão no sentido oposto também é bastante simples. Substitui-se cada dígito octal ou hexadecimal pelo conjunto de 3 ou 4 dígitos binários que o representa.

Exemplo 1.2: converter-os números que seguem para binário.

$$765_8 = 111110101_2$$

$$FED_{16} = 111111101101_2$$

1.2.3 Conversão entre Sistemas Numéricos

Como regra geral, não se pode converter um número representado numa determinada base para outra base simplesmente substituindo-se dígitos de uma base pelos seus equivalentes na outra. Isto funciona somente nos casos em que ambas bases são potências de um mesmo número (como os casos mostrados anteriormente). Quando não é este o caso, será necessário utilizar-se um procedimento mais complexo o qual requer operações aritméticas.. A seguir, será mostrado como converter um número em qualquer base para a base 10, e vice-versa, usando aritmética de base 10.

Como indicado na sub-seção 1.2.1, o valor de um número em qualquer base é representado pela fórmula:

$$D = \sum_{i=-n}^{m-1} d_i \times r^i$$

Onde *r* é a base do número, m indica o número de dígitos à esquerda do ponto e n indica o número de dígitos à direita do ponto.

Uma outra alternativa à equação anterior é re-escrever a parte inteira da equação por:

$$D = ((\ldots((\ d_{m-1}\ )\ r + d_{m-2}\ )\ r + \ldots\ )\ r + d_1\ )\ r + d_0$$

Desta equação, é possível derivar um procedimento iterativo que obtém cada um dos dígitos do número a partir do mais significativo, acumulando a soma decimal ponderada.

Essa equação é também um bom ponto de partida para converter um número decimal D para outra base *r* qualquer. Se dividirmos D por r na equação, a parte parametrizável da equação representa o quociente

$$Q = (\ldots((\ d_{m-1}\ )\ r + d_{m-2}\ )\ r + \ldots\ )\ r + d_1$$

com resto $R = d_0$. Em outras palavras, d_0 é obtido como o resto da divisão inteira de D por *r*. Além disso, como o quociente Q na equação anterior tem o mesmo formato do número original, sabe-se que sucessivas divisões por r permitem obter-se sucessivos dígitos de D, da direita para a esquerda, até que todos os dígitos de D tenham sido encontrados.

1.2.4 Soma e Subtração de Números Binários

O procedimento para a adição e subtração de números binários é semelhante ao que se usa para números decimais. A principal diferença está nas tabelas de adição e subtração, que contêm apenas os dígitos 0 e 1.

Para somar dois números decimais, faz-se a soma de um par de dígitos de cada vez, começando com o dígito menos significativo de cada número. Se a soma de um dado par é igual ou maior que 10, o **excesso** (também chamado de transporte ou ***carry***) é usado na soma do próximo par de dígitos mais significativos. Para somar dois números binários, o procedimento é basicamente o mesmo, mas usa-se um *carry* inicial c_0 igual a 0. A tabela 1.4 mostra a soma s_i e o bit de carry c_{i+1} para cada possível combinação de x_i, y_i e c_i.

Tabela 1.4 - Adição de números binários.

x_i	+ y_i	+ c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Exemplo 1.3: Somar os números binários 987 e 123.

		512	256	128	64	32	16	8	4	2	1
x		1	1	1	1	0	1	1	0	1	1
y					1	1	1	1	0	1	1
carries	1	1	1	1	1	1	1	0	1	1	
x + y	1	0	0	0	1	0	1	0	1	1	0
	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Primeiro se soma $x_0 = 1$ e $y_0 = 1$, produzindo carry $c_1 = 1$ e soma $s_0 = 0$. Em seguida, se soma $x_1 = 1$, $y_1 = 1$ e $c_1 = 1$, obtendo-se *carry* $c_2 = 1$ e soma $s_1 = 1$. Este processo continua até se gerar $c_{11} = 1$ e *carry* $s_{10} = 0$.

A subtração de números binários é realizada de maneira semelhante, subtraindo-se um par de bits, um a cada vez, no entanto, é sempre gerado um bit de empréstimo (*borrow*) e não um bit de *carry*, e um bit de diferença ao invés de um bit de soma. A tabela 1.5 mostra a diferença d_i e o bit de *borrow* b_{i+1} para cada possível combinação de x_i, y_i e b_i.

Tabela 1.5 - Subtração de números binários.

x_i	- y_i	- b_i	d_i	b_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

O procedimento para subtração é muito semelhante ao de adição: a partir dos dígitos menos significativos, gera-se os bits de borrow b_1 e de diferença d_0, de acordo com a Tabela 4, e assim por diante, da direita para a esquerda, até o bit de borrow mais significativo b_m e o bit de diferença mais significativo d_{m-1}.

Exemplo 1.4: Subtrair o número binário 123 de 987.

	512	256	128	64	32	16	8	4	2	1
x	1	1	1	1	0	1	1	0	1	1
y				1	1	1	1	0	1	1
borrows	0	0	1	1	0	0	0	0	0	
x - y	1	1	0	1	1	0	0	0	0	0
	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	d_0

Primeiro faz-se a subtração entre $x_0 = 1$ e $y_0 = 1$, produzindo *borrow* $b_1 = 0$ e diferença $d_0 = 0$. Em seguida, faz-se a subtração de $y_1 = 1$, $b_1 = 1$ de $x_1 = 1$, obtendo-se *borrow* $b_2 = 0$ e soma $s_1 = 0$. Este processo continua até se gerar $d_9 = 1$.

1.2.5 Representação de Números Negativos

Até o presente momento, tratamos somente de números positivos. Números negativos podem ser representados de diversas formas. A representação que usamos normalmente é denominada sinal-magnitude. No entanto, a maioria dos computadores usa o sistema de representação em complemento, para facilitar a implementação dos circuitos aritméticos.

1.2.5.1 Representação de números binários positivos e negativos

Números binários positivos podem ser representados como números sem sinal. No entanto, para se representarem números negativos, é necessária a utilização de alguma

notação. A forma de representação mais simples é **sinal-magnitude**. Nesta representação, o número consiste de duas partes: a **magnitude** e o **sinal**. A magnitude expressa a quantidade e o sinal indica se a quantidade é positiva ou negativa.

Quando sinal-magnitude é usado para números binários, o sinal é representado pelo dígito mais significativo: "0'" indica sinal positivo ao passo que "1" indica sinal negativo. Assim, os números +9 e –9 escritos em binário se diferenciam somente pelo bit mais significativo:

$$+9 = 01001$$

$$-9 = 11001$$

Note que foram necessários 5 bits para representar esses números: 4 para a magnitude e 1 (o mais da esquerda) para representar o sinal.

Dado um número binário com n bits, o intervalo de representação possível será de $-(2^{n-1}-1)$ até $+(2^{n-1}-1)$. Note também que há duas representações possíveis para o zero: -0 e +0.

1.2.5.2 Sistema de representação em complementos

A representação em complemento foi criada com o intuito de simplificar a operação de subtração e manipulações lógicas, evitando a necessidade de comparações de magnitude e sinal.

Dado um sistema de base *r* qualquer, existem dois tipos de complementos possíveis:

- Complemento de *r*
- Complemento de (*r*-1).

Iniciemos o estudo pelo **complemento de (*r*-1)**. Seja um número N na base *r* representado com n dígitos. O complemento de (*r*-1) de N é definido por $(r^n-1) - N$. Para números decimais, $r = 10$, $(r-1) = 9$ e o complemento de 9 de N é dado por $(10^n-1) - N$.

Exemplo 1.5: encontrar o complemento de 9 dos números decimais que seguem.

987_{10}: $(1000 - 1) - 987 = 999 - 987 = 12$

546700_{10}:

12389_{10}:

No caso de N ser um número binário, $r = 2$, $(r-1) = 1$ e o complemento de 1 de N é dado por $(2^n-1) - N$.

Exemplo 1.6: encontrar o complemento de 1 dos números binários que seguem.

10111_2: $(32 - 1) - 23 = 31 - 23 = 8 = 1000_2$

1011001_2:

0001111_2:

Já o **complemento de *r*** de um número N na base *r* representado com n dígitos é definido por $r^n - N$, se $N \neq 0$ e 0 se N = 0. Comparando-se com o complemento de (*r*-1), notamos que o complemento de *r* é obtido somando-se 1 ao complemento de (*r*-1). Para números decimais, *r* = 10, e o complemento de 10 de N é dado por $10^n - N$.

O complemento de r de um número binário é chamado **complemento de dois** e a representação derivada dele é chamada **representação em complemento de dois**.

Para números binários, o sinal do número é representado pelo bit mais significativo, que é 0 para números positivos e 1 para negativos. Assim, pode-se obter um número negativo a partir de um número positivo, complementando-se cada bit (i.e., substituindo-se cada 0 por 1 e cada 1 por 0), incluindo o bit de sinal e somando-se 1. O bit de *carry* que ocorra deve ser ignorado. Portanto, quando se complementa o 0 e o bit de *carry* mais significativo é descartado, obtém-se 0 como complemento de dois. Isto mostra que o 0 tem uma representação única em complemento de dois. Como o 0 é um número positivo, se terá um número positivo não-zero a menos do que números negativos, ou seja, o intervalo de números que podem ser representados em complemento de dois é $-(2^{n-1})$ até $+(2^{n-1} - 1)$.

Exemplo 1.7: determine o intervalo de representação para números binários com 4 dígitos (bits), assumindo-se o uso de complemento de 2

Pelas fórmulas acima, conclui-se que o intervalo de representação para números de quatro bits em complemento de 2 vai de $-2^3 = -8$ até $2^3 - 1 = +7$.

Uma maneira simples de se obter o complemento de dois de um número binário é fazer o complemento do número e somar 1 a ele.

Exemplo 1.8: encontre o complemento de 2 do número binário que segue.

10111_2: $01000_2 + 1_2 = 01001_2$

1.2.5.3 Adição e subtração em complemento de dois

Como dito anteriormente, a representação em complemento de dois foi desenvolvida para facilitar a implementação de adição e subtração de números binários. Com o complemento de dois é possível realizar somas e subtrações de números com e sem sinal **usando o mesmo circuito lógico**.

Para somar números em complemento de dois, usa-se a aritmética binária mostrada na tabela 1.4 e ignora-se qualquer bit de *carry* além do bit de sinal. Por exemplo, somar dois números positivos gera um resultado positivo correto:

```
   0010 (+2)
+  0100 (+4)
-------------
   0110 (+6)
```

Da mesma maneira, somar dois números negativos sempre produzirá uma soma negativa correta, desde que seja ignorado o bit de *carry* além do bit de sinal:

```
   1110 (-2)
+  1100 (-4)
-------------
   1010 (-6)
```

No entanto, existem casos em que uma operação produz um resultado que excede o intervalo de representação do sistema numérico, criando uma condição chamada **estouro** ou ***overflow***.

Via de regra, a adição de dois números com sinais diferentes nunca produz um *overflow*. Entretanto, quando se somam dois números com o mesmo sinal que produz uma soma que é maior que o maior número representável, pode-se obter um resultado incorreto. Por exemplo,

```
   0100 (+4)
+  0101 (+5)
-------------
   1001 (-7)
```

Do mesmo modo,

```
   1100 (-4)
+  1011 ( -5)
-------------
   0111 (+7)
```

Como os exemplos sugerem, é possível identificar uma regra simples para detectar *overflow* em adições: ocorre um *overflow* sempre que o sinal da soma é diferente do sinal dos operandos. Mas, como será visto, para implementar um circuito implementando a soma, é mais fácil identificar *oveflow* usando a seguinte regra: ocorre um *overflow* sempre que os bits de *carry* para o bit de sinal e a partir do bit de sinal são diferentes.

Para realizar a subtração em complemento de dois, pode-se usar o mesmo procedimento usado para números binários sem sinal e as regras para a detecção de *overflow* são as mesmas da adição. Mas, para implementar a subtração não se usa este procedimento. Ao invés disto, faz-se o complemento de dois do número que se quer subtrair (subtraendo) e faz-se a soma dele com o número do qual vai se subtrair. Por exemplo,

```
   0010 (+2)
+  1100          Complemento de dois de (+4)
   000           Carries
-------------
   1110 ( -2)    Em complemento de dois
```

Para se implementar esta subtração basta um circuito complementador e um somador. Como se está realizando uma adição, a regra para detecção de *overflow* é a da adição.

Exercícios

Exercício 1.1 - Realizar as conversões indicadas a seguir.

a) 179 para binário

179 ÷ 2 = 89 resto = 1 (dígito menos significativo)

89 ÷ 2 = 44 resto = 1

b) 467 para octal

467 ÷ 8 = 58 resto = 3 (dígito menos significativo)

58 ÷ 8 =

c) 3417 para hexadecimal

3417 ÷ 16 =

Exercício 1.2 - Encontrar a representação em complemento de *r* para os exemplos anteriores.

987_{10} : 1000 - 987 = 13

546700_{10}:

12389_{10}:

10111_2: 32 - 23 = 9 = 1001_2

1011001_2:

0001111_2:

Exercício 1.3 - Encontrar, para os exemplos anteriores, a representação em complemento de 2 usando operações de complemento.

1011001_2:

0001111_2:

Bibliografia Suplementar

[1] GAJSKI, Daniel D. **Principles of Digital Design**, New Jersey: Prentice Hall, 1997 (ISBN 0-13-301144-5)

[2] MANO, M. Morris; **Computer Engineering: Hardware Design.** New Jersey: Prentice Hall, 1988 (ISBN 0-13-162926-3)