Arquitetura e Organização de Computadores

Prof. Marco Câmara
V4.5

Índice

1 ENGENHARIA DE SOFTWARE & ARQUITETURA DE COMPUTADORES 5

2 DIGITAL X ANALÓGICO 6

3 REPRESENTAÇÃO DE INFORMAÇÕES NUMÉRICAS 7

3.1 ALGARISMOS ROMANOS 8

3.2 UNIDADES DE MEDIDA BASEADAS NO CORPO HUMANO 8

4 SISTEMAS DE NUMERAÇÃO 9

4.1 ALGARISMOS 9

4.2 O CONCEITO DE BASE 9

4.3 NOTAÇÃO POSICIONAL 9

4.4 POSIÇÃO DE UM ALGARISMO 10

5 UTILIZANDO MAIS DE UM SISTEMA DE NUMERAÇÃO 10

5.1 BASE 2 (BINÁRIA) 12

5.2 UNIDADES DE ARMAZENAMENTO BINÁRIAS 12

5.3 BASES 8 (OCTAL) E 16 (HEXADECIMAL) 13

5.4 CONVERSÃO ENTRE BASES 13

5.5 CONJUNTOS NUMÉRICOS 14

5.6 NÚMEROS NEGATIVOS 14

5.7 NÚMEROS FRACIONÁRIOS E DÍZIMAS 15

6 TABELA ASCII 16

7 COMPUTADOR – UMA MÁQUINA DE INFORMAÇÕES 18

7.1 O ÁBACO 18

7.2 A MÁQUINA DE ANTICÍTERA 18

7.3 A MÁQUINA ANALÍTICA 18

7.4 A MÁQUINA DE TURING 19

7.5 ENIAC 19

7.6 O IAS, OU MÁQUINA DE VON NEUMANN 19

7.7 A TROCA DOS RELÊS PELAS VÁLVULAS 19

7.8 A TROCA DAS VÁLVULAS PELOS TRANSISTORES 20

7.9 O SURGIMENTO DOS CIRCUITOS INTEGRADOS 20

8 CIRCUITOS ELÉTRICOS 20

9 FUNÇÕES LÓGICAS BINÁRIAS EM CIRCUITOS ELÉTRICOS 22

9.1 LÓGICA E (AND) 22

9.2 LÓGICA OU (OR) 23

9.3 LÓGICA NÃO (NOT) 23

10 EVOLUÇÃO DA ELETRÔNICA NAS "GERAÇÕES" DE COMPUTADORES 24

11 PORTAS LÓGICAS BÁSICAS 25

11.1 PORTA AND (E) 25

11.2 PORTA OR (OU) 25

11.3 PORTA NOT (NÃO) 26

11.4 PORTA XOR (OU EXCLUSIVO) 26

12 UM EXEMPLO DE CIRCUITO ARITMÉTICO 27

13 SÍNTESE DE CIRCUITO LÓGICO A PARTIR DA TABELA-VERDADE 29

Prof. Marco Câmara maccamara@gmail.com
Home Page: www.logicengenharia.com.br/mcamara

Índice

14 SIMPLIFICAÇÃO DE CIRCUITOS LÓGICOS 31

15 LÓGICA COMBINACIONAL X LÓGICA SEQUENCIAL 33

16 ESTRUTURA DE UM COMPUTADOR 33
16.1 ENIAC: O PRIMEIRO COMPUTADOR DE USO GERAL 33
16.2 O IAS, A MÁQUINA DE VON NEUMANN 33
16.3 ESTRUTURA DE ALTO NÍVEL 35
16.4 ESTRUTURA DA CPU 35
16.5 ESTRUTURA DA UNIDADE DE CONTROLE 35

17 OPERANDO O IAS 35
17.1 OS REGISTRADORES DO IAS 36
17.2 CICLO DE INSTRUÇÃO NO IAS 36
17.3 O *ASSEMBLY* DO IAS 37

18 MEMÓRIA – CARACTERÍSTICAS 38
18.1 LOCALIZAÇÃO 38
18.2 CAPACIDADE 39
18.3 UNIDADE DE TRANSFERÊNCIA 39
18.4 MÉTODO DE ACESSO 39
18.5 DESEMPENHO 39
18.6 TIPO FÍSICO 39
18.7 CARACTERÍSTICAS FÍSICAS 39
18.8 ORGANIZAÇÃO 40

19 HIERARQUIA DE MEMÓRIA 40

20 A MEMÓRIA CACHE 41

21 BARRAMENTOS 42

22 MÁQUINAS PARALELAS 43
22.1 SISD 43
22.2 SIMD 44
22.3 MISD 44
22.4 MIMD 44
22.5 ARQUITETURA *PIPELINE* 44
22.6 ARQUITETURA SUPERESCALAR 45

23 COMPUTAÇÃO FÍSICA 45
23.1 MICROCONTROLADORES X MICROPROCESSADORES 46
23.2 APLICAÇÕES TÍPICAS DA COMPUTAÇÃO FÍSICA 46

24 *HARDWARE* PARA COMPUTAÇÃO FÍSICA 47
24.1 MICROCONTROLADORES 47
24.2 PLACAS DE PROTÓTIPO 48
24.3 SBC (SINGLE BOARD COMPUTERS) 48

25 O ARDUÍNO UNO 49
25.1 PORTAS DIGITAIS 49
25.2 PORTAS D0 E D1: 50
25.3 PORTAS D2 E D3: 51
25.4 PORTAS D10 A D13: 51
25.5 PORTA 13: 51
25.6 PORTAS ANALÓGICAS 51

25.7 PORTAS A4 E A5 52
25.8 CONEXÕES AO ARDUÍNO UNO 53
25.9 POR DENTRO DO ARDUÍNO UNO R3 53
26 OUTRAS PLACAS ARDUÍNO 54
27 OUTRAS PLACAS DE PROTÓTIPO 55
28 PROGRAMANDO UM MICROCONTROLADOR 56
28.1 LINGUAGEM DE PROGRAMAÇÃO E SEÇÕES OBRIGATÓRIAS 57
28.2 CARGA E EXECUÇÃO DO *SKETCH* 57
28.3 UM EXEMPLO DE *SKETCH* 58
28.4 ESTRUTURA DA LINGUAGEM E SINTAXE 60
28.5 ALGUMAS LIMITAÇÕES IMPORTANTES 61
29 SIMULANDO PROJETOS COM PLACAS DE PROTÓTIPO 61
30 REFERÊNCIAS 63

1 Engenharia de Software & Arquitetura de Computadores

Por que estudar Arquitetura e Organização de Computadores em um curso como Engenharia de *Software*, ou Análise de Sistemas?

Software rodam em *hardware*, e as características específicas do *hardware* podem influenciar diversos aspectos do *software*, como funcionalidade, performance, facilidade de uso, confiabilidade, capilaridade, interações com o ambiente, dispositivos e usuários, custo, viabilidade e complexidade.

Funcionalidade: é possível rodar o Waze em um notebook convencional? Por que não? Falta *hardware* (receptor GPS).

Capilaridade: vale a pena desenvolver uma aplicação de rede social que só funcione bem em celulares de primeira linha como o Galaxy S24 UItra 5G, o Pixel 8 Pro, o Huawei Pura 70 Ultra ou o iPhone 15ProMax)? Não, não atingiria o público desejado – aplicações precisam ser compatíveis com *hardware* de baixo custo.

Interações com o Ambiente: dá para pedir para esquentar a água para o banho e ligar o ar-condicionado do quarto antes de chegar em casa, pelo celular? Sim, podemos, mas precisaremos ter *hardware* específico em casa para transformar os cliques no aplicativo na ativação de dispositivos elétricos em casa.

Interações com Dispositivos: é possível desbloquear a tela de entrada do Windows do seu computador de casa tocando com o polegar na tela do seu celular? Sim, mas é necessário não só um leitor seguro e eficaz de digitais no celular, como também uma rede de comunicação rápida, segura, e de pequena distância para autenticar o usuário no seu computador.

Interações com o Usuário: dá para utilizar um celular como controle remoto de uma console de videogame? Claro, mas é necessário que o celular possua um acelerômetro, e um chip controlador que permitam transformar, com precisão, a posição do celular em coordenadas de entrada para o jogo.

Custo: é possível implementar a "internet das coisas" em "coisas baratas"? Claro, mas precisamos simplificar, reduzir o tamanho e o consumo de dispositivos computacionais com capacidade de comunicação para que eletrodomésticos de baixo custo possam se comunicar com a internet.

Viabilidade: seria possível alertar o serviço médico de urgência quando um idoso sofrer uma queda usando o seu *smart watch*? É possível sim, mas precisamos implementar os recursos e sensores de um celular em um dispositivo muito menor, com bateria pequena e que tenha uma conexão muito confiável à internet.

Complexidade: sabia que provavelmente é mais barato colocar um computador para controlar um semáforo, do que desenvolver um circuito eletrônico equivalente? Sim, normalmente é, pois os avanços no desenvolvimento de componentes padronizados para computadores reduziram muito a complexidade e o custo na montagem de computadores, tornando os mesmos onipresentes em aplicações diversas.

2 Digital x Analógico

Na história humana, o "trabalho" na maior parte do tempo empregava a força dos músculos. Aos poucos, o homem começou a desenvolver ferramentas, e a aproveitar elementos naturais para ajudá-lo (o curso da água, o vento e o fogo, por exemplo). Também foi usada a força animal.

As máquinas sempre existiram, mas elas começaram a substituir este tipo de trabalho humano de forma mais extensiva a partir da Revolução Industrial.

A substituição do trabalho intelectual, por outro lado, sempre foi muito rara, sendo normalmente "terceirizada" para outros humanos. No último século, as máquinas começaram a fazer parte do trabalho intelectual humano. Alguns inclusive acreditam que um dia elas farão TODO o trabalho humano.

Os computadores são aplicados majoritariamente no trabalho intelectual, e também para aprimorar as máquinas que executam trabalhos físicos. Para isso eles precisam capturar, armazenar, processar e fornecer uma matéria prima – ou produto – bem específico: a informação.

A informação normalmente não tem existência física; ela pode, no máximo, ser "representada" fisicamente. Livros, fotografias, DVDs e *pendrives* não são informações, e sim "mídias" que contêm as mesmas.

Informações podem ser de dois tipos: analógicas ou digitais.

As informações digitais possuem valores discretos, ou seja, entre dois valores possíveis de uma informação digital, existem **finitos** valores intermediários. Por exemplo, ao se pesar em uma moderna balança de uma farmácia com 3 dígitos, se duas pessoas diferentes tiverem pesos de 75,1 e 75,2Kg, respectivamente, não existirá nenhuma pessoa com peso intermediário (75,12Kg, por exemplo) que consiga provar isso usando aquela balança digital.

Por outro lado, se a farmácia tivesse uma balança analógica daquelas antigas, existiriam infinitas variações possíveis de peso entre as duas pessoas do exemplo anterior – muito embora provavelmente seria muito complicado identificar a diferença de peso em alguns casos.

O nosso mundo está cheio de informações analógicas. Basicamente, a maior parte das grandezas, como massa, velocidade e temperatura, por exemplo, é analógica. Apenas as contagens, e algumas informações específicas como data, gênero e estado civil, por exemplo, são digitais.

Existem diferenças marcantes na dificuldade que temos ao lidar com informações digitais ou analógicas. Tente, por exemplo, repassar, à distância, para uma amiga, qual a cor da camisa que você deseja que ela compre para você em um *shopping*. A cor de uma peça de roupa é uma informação analógica, com infinitas variações, e se você for detalhista, será simplesmente impossível garantir que a cor será transmitida com exatidão. Indicar o valor que deve ser sacado em um caixa eletrônico, por outro lado, não deixa margem a dúvidas. O valor a ser sacado é uma grandeza digital, e independente do fato de você ser, ou não, detalhista, sempre existirá uma forma

precisa, e à prova de erros, para transmitir esta informação. Esta mesma dificuldade vale tanto para a transmissão de informações (que é estudada nos livros de Redes de Computadores), como para armazenar, interpretar e fazer operações envolvendo estas informações.

A complexidade de representação, armazenamento e processamento de informações analógicas levou à padronização das informações digitais nas tecnologias mais modernas. Hoje usar uma informação analógica é, muitas vezes, considerado como algo "ultrapassado". É importante, no entanto, fazer justiça para pelo menos uma vantagem das informações analógicas – o ser humano tem muita facilidade para interpretá-las rapidamente. Este é o motivo pelo qual os painéis de instrumentos das aeronaves mais sofisticadas, por exemplo, apresentarem "ponteiros" analógicos, mesmo que simulados, ao invés de valores digitais. O mesmo vale para a maior parte dos velocímetros dos automóveis, por exemplo.

Os dispositivos computacionais, embora possam lidar com informações analógicas, normalmente precisam convertê-las para o formato digital ao recebê-las, e algumas vezes precisam convertê-las de volta para o formato analógico ao fornecê-las para o mundo externo e seus usuários.

3 Representação de Informações Numéricas

Desde o início da caminhada do ser humano no nosso planeta, sempre foi necessário lidar com a representação de informações, muitas vezes digitais. Vamos usar como exemplo a contagem:

- Quantas cabeças de gado um criador possuía?

- Quantas "luas" durará uma viagem?

- Quantos guerreiros a tribo adversária possui?

Sem uma forma de representar estas informações, o homem primitivo apenas associava a contagem a uma avaliação qualitativa. Assim, um criador possuía "muitas" ovelhas; a viagem durava "poucas luas"; ou a tribo adversária possuía "muitos guerreiros".

Demorou muito, mas quando o homem começou a se organizar em grupos e depois em civilizações mais complexas, começaram a surgir métodos de representação cada vez mais elaborados para estas contagens. Associar a contagem a um número específicos de dedos da mão, por exemplo, funcionava bem para quantidades pequenas (até hoje funciona para uma criança pequena quando perguntamos a sua idade). No entanto, para quantidades maiores, cada vez mais comuns nos agrupamentos humanos mais avançados, foi necessário criar um esquema de representação mais evoluído.

Temos alguns exemplos que funcionaram temporariamente – alguns até hoje – mas que tendem a ser substituídos devido à sua ineficiência:

3.1 Algarismos Romanos

Até hoje utilizados para algumas funções específicas, os algarismos romanos, base de um dos maiores impérios que o mundo conheceu, associavam algumas letras do alfabeto a quantidades específicas. Para identificar quantidades diferentes, utilizava-se a justaposição de outras letras à esquerda (para diminuir), ou à direita (para aumentar).

Embora tenha funcionado, você já tentou fazer operações simples, como uma soma, usando algarismos romanos? Neste vídeo disponível no Youtube um professor demonstra o processo.

(https://www.youtube.com/watch?v=OXLVL_TiMAk#t=03m24s).

3.2 Unidades de Medida baseadas no corpo humano

Você deve conhecer, ou ao menos ter ouvido falar, de unidades de medida baseadas em partes do corpo humano. Talvez já tenha até tentado medir distâncias usando estas unidades. Veja algumas:

- O cúbito, usado pelos egípcios, era a distância entre o cotovelo e a ponta do dedo médio do faraó (52,4cm). A medida mudava em outras regiões, como na Suméria, onde era 39,5cm e na Assíria, onde era 54,9cm;
- A **milha** (*mile*), utilizada na Inglaterra e nos EUA, e mundialmente em algumas medições específicas, surgiu da medida de 1000 passos de um centurião romano, o que equivaleria a 1.609,34m;
- A **jarda** (yard), usada em medidas esportivas na Inglaterra e nos EUA, é a distância entre o nariz e a ponta do polegar com braço esticado do rei Henrique I. Ela equivale a 3 pés, ou 91,44cm;
- O **pé** (*foot*), largamente utilizado na Inglaterra e EUA, é utilizada mundialmente na aviação, e equivale 12 polegadas, ou 30,48cm.
- A **polegada** (*inch*), largamente utilizada na Inglaterra e EUA, é mundialmente utilizada em algumas medições específicas, como em tubulações para construção civil, por exemplo. Equivale a 2,54cm.

Embora algumas destas unidades sejam utilizadas até hoje, imagine a dificuldade para operações com múltiplos e submúltiplos. Quantos pés tem uma milha, por exemplo?

Não só por isso, como também por uma questão de padronização mundial, estas unidades vêm sendo, aos poucos, substituídas pelas equivalentes no SI (Sistema Internacional de medidas), baseado no metro, seus múltiplos e submúltiplos.

A verdade é que a utilização de sistemas de medidas diferentes ao redor do mundo gera e continuará gerando problemas. Um dos casos mais famosos ocorreu com a NASA, que no final de 1988 perdeu uma sonda de centenas de milhões de dólares, e mais de uma década de projeto e desenvolvimento devido a um problema de conversão de unidades. A MCO (*Mars Climate Orbiter*) se estatelou na superfície de marte antes de fazer quaisquer medições ou testes para a qual ela foi projetada.

Para nossa sorte, no entanto, não herdamos o sistema de numeração da civilização romana, e o sistema métrico universal vem aos poucos substituindo os sistemas antigos.

O sistema em uso hoje em todo o mundo é o Sistema Numérico Indo-Arábico. Este sistema baseia-se em dez dígitos (por analogia aos 10 dedos das mãos) com símbolos aparentemente desenvolvidos a partir de letras utilizadas nas regiões ocidentais no mundo árabe. A palavra “algarismo”, por exemplo, tem sua raiz no nome do célebre matemático árabe Al-Khwarizmi. O matemático italiano Leonardo Fibonacci introduziu o conceito na Europa no século XII, de onde acabou por se tornar o padrão mundialmente adotado hoje.

4 Sistemas de numeração

O conjunto de definições adotadas hoje em dia para representar números, como vimos, baseia-se em 10 algarismos diferentes, e algumas regras de representação que conhecemos quase que institivamente, devido ao convívio diário com a representação de quantidades e valores.

Este conjunto de definições recebe o nome de Sistema de Numeração. Vamos entender a seguir os seus elementos.

4.1 Algarismos

Utilizamos 10 símbolos indo-arábicos para representar quantidades (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9). Estes símbolos se estabeleceram por questões culturais e históricas, embora existam lendas modernas que tentam atribuir as quantidades representadas ao desenho dos símbolos (número de ângulos).

4.2 O conceito de Base

A quantidade de símbolos existentes em um sistema de numeração é chamada de **base** do sistema de numeração. No sistema que adotamos hoje, temos 10 símbolos, o que provavelmente está associado à quantidade de dedos nas mãos humanas. É por isto que cada símbolo representa uma quantidade entre zero e nove.

Quando agrupamos diversos símbolos, conseguimos representar quantidades bem maiores, ou menores. Para isto utiliza-se a notação posicional.

4.3 Notação Posicional

O sistema de numeração indo-arábico não só define os símbolos dos algarismos, como também utiliza o conceito de Notação Posicional. Isto o torna muito mais simples do que os sistemas anteriores. A notação posicional determina que, em um número completo contendo diversos dígitos, a quantidade representada por cada dígito está relacionada não só ao símbolo utilizado, como também à sua posição dentro do número. No final, a soma das quantidades representadas por cada um dos algarismos determina a quantidade representada pelo número.

Este conceito vale para todas as bases. Logo, ele também permite a conversão entre números representados em diferentes bases de numeração.

Para explicar melhor, vamos recordar um conceito que você certamente deve ter visto nos primeiros anos de sua formação em matemática. Imagine um valor específico, como R$ 125,40 (Cento e vinte e cinco reais e quarenta centavos). Como interpretamos este valor ao lê-lo?

Cada algarismo no número tem um valor que depende de sua posição:

O primeiro algarismo (1) representa uma centena.

O segundo (2) representa duas dezenas.

O terceiro (5) representa cinco unidades.

Veja que cada algarismo representa um valor que corresponde ao valor do algarismo daquela posição (1, 2 e 5), multiplicado pelo valor que aquela posição representa (centenas, dezenas e unidades).

Se continuarmos, o primeiro algarismo à direita de vírgula (4) também representa o seu valor individual, multiplicado pelo valor da posição (dezenas de centavos).

O valor total é obtido somando os valores representados por cada algarismo em sua respectiva posição, ou seja, cento e vinte e cinco reais e quarenta centavos.

Esta forma de interpretar um número composto por diversos algarismos é chamada de **notação posicional**, e funciona para qualquer base de numeração, e não só para a base decimal.

O que muda, então, de uma base para outra, são duas informações: o número de algarismos do sistema, ou base, e o valor de cada posição.

4.4 Posição de um algarismo

Para entender o valor de cada posição, primeiro precisamos entender como as posições são identificadas:

Veja que a posição é contada a partir de zero, da vírgula para a esquerda, e é incrementada a cada deslocamento para a esquerda. No caso de deslocamento para a direita (pulando a vírgula), a posição é decrementada, ficando negativa.

Sabendo o valor da posição, basta elevar a base do sistema ao número da posição. Neste exemplo:

O algarismo 1 na posição 2 representa $1 \times 10^2 = 100$;

O algarismo 2 na posição 1 representa $2 \times 10^1 = 20$;

O algarismo 5 na posição 0 representa $5 \times 10^0 = 5$;

O algarismo 4 na posição -1 representa $4 \times 10^{-1} = 0{,}4$.

Generalizando, podemos dizer:

Valor do número = $\sum$ Valores de Posição, onde:

Valor de Posição = $V_a \times Base^p$

V_a = Valor do algarismo

p = Posição

5 Utilizando mais de um Sistema de Numeração

Desde crianças, aprendemos a representar as quantidades através do sistema indo-arábico com 10 símbolos, também chamado de **sistema decimal**, ou de **base 10**. Por isso associamos rapidamente as combinações dos algarismos dentro dos números às respectivas quantidades representadas. Ao ver um produto em uma loja, por exemplo, basta ler o preço impresso para concluir se está caro, barato, comparar produtos diferentes etc.

No entanto, às vezes sem perceber, lidamos no dia a dia com outro sistema de numeração, o nosso sistema de marcação de tempo. Embora utilize os mesmos símbolos do sistema decimal, a base não é 10. Temos 60 segundos em um minuto e 60 minutos em uma hora. A quantidade de valores diferentes é 60, e não 10.

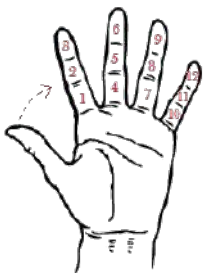


Figura 1 - A dúzia nos dedos

Trata-se de um sistema sexagesimal. Acredita-se que este sistema foi criado pelos sumérios por volta de 2.000 AC. Eles consideravam 60 um número mágico, por ser o menor número divisível simultaneamente por 1, 2, 3, 4, 5 e 6. Eles também utilizam a dúzia, resultado da contagem das divisões dos

dedos da mão (excluindo o polegar), e que também é divisão de 60 por 5. A dúzia era utilizada para dividir a parte clara e a parte escura de cada dia, o que resultou no uso das 12 horas. Temos, neste caso, um sistema duodecimal.

Figura 2 - Christian Huygens

Como não existiam instrumentos precisos o suficiente para dividir a hora, acredita-se que apenas em 1670 surgiu o "minutus", que em latim significa "pequeno". Este seria resultado do trabalho do cientista holandês Christian Huygens. Huygens tem, entre outros trabalhos de destaque, a patente do primeiro relógio de pêndulo, extremamente preciso para a época.

Por volta de 1800, surgiu a necessidade de dividir o tempo em intervalos ainda menores, com a utilização do "pequeno de segunda ordem", que hoje é chamado apenas de segundo. Daí chegamos ao nosso sistema de divisão do tempo.

A divisão do tempo em dias, semanas, meses e anos também tem origem histórica bem interessante, mas que foge dos objetivos de nossa disciplina.

O que vemos é que os sistemas de numeração são estabelecidos por motivos culturais, históricos e até anatômicos! Se estivéssemos estudando uma suposta civilização alienígena com oito dedos, por exemplo, seguindo o mesmo raciocínio, e considerando as coincidências impossíveis deles adotarem o mesmo sistema posicional, e até os mesmos símbolos (!?!?), existiria uma grande chance de eles adotarem a base 8. Neste caso, como ocorreria, por exemplo, uma contagem nesta base?

0, 1, 2, 3, 4, 5, 6, 7 ...

Após o sete, eles não teriam mais algarismos, logo teriam que seguir o mesmo procedimento, e começar a escrever números com dois dígitos:

10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22 ... 77 ...

Quando chegassem a 77, o processo se repetiria, mas agora com 3 dígitos:

100, 101, 102 ...

A diferença, neste caso, é que o número 102, por exemplo, não representaria uma centena e duas unidades, e sim uma quantidade diferente. Mais tarde veremos como poderíamos fazer esta conversão para a base decimal.

Ainda discutindo o conceito de base, mas agora dentro do foco do nosso curso, temos que analisar alguns sistemas de numeração que são particularmente interessantes quando falamos de computadores. O que muda neste caso é **a base do sistema**, ou seja, a quantidade de símbolos utilizados. E assim como no caso da adoção de informações digitais ao invés de analógicas, novamente o argumento é a simplicidade.

O sistema decimal exige a identificação de 10 diferentes símbolos, que precisam ser representados, dentro de um computador, de formas diferentes. Quanto maior a quantidade de símbolos diferentes a serem identificados, obviamente maior é possibilidade de falhas, já que símbolos diferentes podem ser confundidos. Além disto, quando você tiver oportunidade de se aprofundar nos métodos utilizados pelos computadores para realizar contagens, operações matemáticas e comparações, perceberá que uma quantidade menor de símbolos fatalmente aumenta a simplicidade operacional.

5.1 Base 2 (Binária)

De todas as bases possíveis, a menor é a binária (dois símbolos, 0 e 1). Logo, esta é a mais simples de todas. Por este motivo ela foi a base adotada nos dispositivos computacionais. Nos computadores convencionais, a representação dos dígitos binários é feita através de sinais elétricos. Graças à existência de apenas dois valores válidos, se reduz muito a probabilidade de erros relacionados à troca de dígitos em um sistema computacional corretamente implementado.

Com apenas dois símbolos, a contagem é bem simples, embora números grandes acabem exigindo muitos dígitos. Para um ser humano, parece complicado manipular grandes quantidades de dígitos. Este é o motivo principal da utilização das bases 8 e 16 para representação de informações internas do computador, que veremos em seguida.

Em termos físicos, considerando o armazenamento de informações dentro de um componente lógico baseado na tecnologia CMOS, tensões elétricas com valores entre 0 a 1/3 da tensão de alimentação simbolizam o valor 0; e de 2/3 até a tensão de alimentação, simbolizam o valor 1.

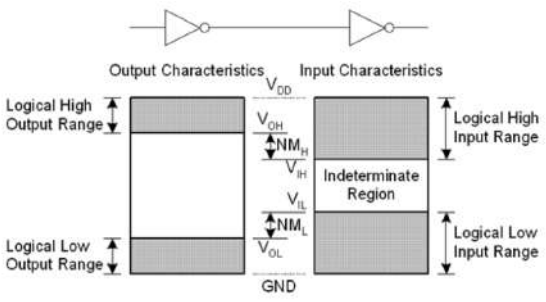


Figura 3 - Representação Elétrica dos bits

A figura ao lado demonstra a robustez deste esquema de codificação. Existe uma grande faixa de valores na saída, que podem ser identificados como 1 ou 0. Esta faixa é ainda maior na leitura de informações binárias, reduzindo muito a propagação de erros causados por problemas elétricos, por exemplo.

5.2 Unidades de Armazenamento Binárias

Cada dígito binário é conhecido como bit (*Binary digIT*, uma sigla estranha para nós, porém comum na cultura americana). Conjuntos ou múltiplos de bits têm nomes específicos. Como se trata de uma base binária, os múltiplos são sempre potências de 2:

4 bits (2^2) = 1 nibble

8 bits (2^3) = 1 Byte.

O Byte (utiliza-se inicial maiúscula, inclusive nas siglas, para diferenciar de bit), por sua vez, também tem seus múltiplos com base em potências de 2:

1024 Bytes (2^{10} Bytes) = 1 KiloByte

1024 KB (2^{20} Bytes) = 1 MegaByte

1024 MB (2^{30} Bytes) = 1 GigaByte

1024 GB (2^{40} Bytes) = 1 TeraByte

1024 TB (2^{50} Bytes) = 1 PetaByte

1024 PB (2^{60} Bytes) = 1 ExaByte

5.3 Bases 8 (Octal) e 16 (Hexadecimal)

Eventualmente, dispositivos computacionais utilizam também outras bases, como a octal (8 símbolos) e a hexadecimal (16 símbolos). Em ambos os casos, temos uma potência de 2 ($8=2^3$ e $16=2^4$), já que o objetivo é utilizar uma base cujos algarismos representem conjuntos de bits, mas que possuam menos dígitos, ou seja, uma representação simplificada, para análise humana, de conjuntos de algarismos binários.

A base 8 permite representar, portanto, 3 dígitos binários com apenas um dígito octal. Já a base 16, mais comumente utilizada, permite representar 4 dígitos. Utilizam-se os mesmos símbolos já conhecidos da base 10 para ambos os sistemas. No caso da base hexadecimal, como são necessários 16 símbolos, acrescentam-se as letras de A a F para representar as quantidades de 10 a 15, respectivamente. É comum utilizar-se um sufixo "h" para identificar que se trata de um número hexadecimal, muito embora a notação matemática para bases diferentes de 10 é a colocação do número entre parênteses, com a base identificada ao lado direito em subscrito. Sendo assim, a quantidade trinta pode ser representada em hexadecimal por 1Dh, ou $(1D)_{16}$.

5.4 Conversão entre Bases

É importante destacar que, independente da base utilizada, uma representação numérica serve apenas para identificar uma informação. Sendo assim, uma quantidade, por exemplo, pode ser representada de diferentes formas, por diferentes números nas bases 2, 8 e 16. Todos elas representam a mesma quantidade.

A conversão pode ser feita pela aplicação direta dos conceitos de notação posicional. Vamos ver alguns exemplos?

Para começar, vamos converter um número da base 8 para a base 10. Que quantidade o número $(117)_8$ representa na base 10? Aplicando a notação posicional, temos:

$1 \times 8^2 + 1 \times 8^1 + 7 \times 8^0 =$

$1 \times 64 + 1 \times 8 + 7 \times 1 =$

$64 + 8 + 7 = 79$

Logo, $(117)_8$ é igual a $(79)_{10}$.

Vamos para outro exemplo. Agora vamos converter o número 62 na base 10 para a base 8. O primeiro passo é descobrir quantos dígitos terá o número na base 8. Para isso, calculamos as potências de 8 para identificar qual a maior posição que precisará ser representada. Como o número 62 tem dois dígitos, certamente na base 8 ele terá no mínimo a mesma quantidade de dígitos, já que a quantidade de dígitos aumenta com bases menores. Vamos, então, começar com três dígitos, ou seja, o dígito mais significativo estaria na posição 2:

$8^2 = 64$; 62 é menor que 64, logo não há dígito na posição 2.

Vamos então tentar a posição 1.

$8^1 = 8$; 62 é maior que 8.

Logo, vamos procurar o maior múltiplo de 8 que seja menor que 62:

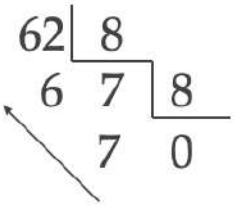
$8 \times 7 = 56 \Rightarrow$ 7 é o algarismo na posição 1.

Para chegarmos aos 62, ainda faltam 6. Isso a gente resolve no próximo dígito.

$8^0 = 1$; $6 = 1 \times 6 \Rightarrow$ 6 é o algarismo na posição 0

Logo, $(62)_{10}$ é igual a $(76)_8$.

Veja que aplicando os conceitos da notação posicional, é perfeitamente possível converter números entre bases diferentes. No entanto, existem técnicas mais simples e rápidas para conversão de números entre bases. Mesmo elas, no entanto, também se baseiam no conceito de notação posicional.



Uma das técnicas mais conhecidas é a técnica das divisões sucessivas, utilizada para converter números na base 10 para qualquer outra base. Vamos testá-la repetindo o último exemplo?

Veja que basta dividir o número na base 10 pela base desejada sucessivamente, até obter zero como quociente.

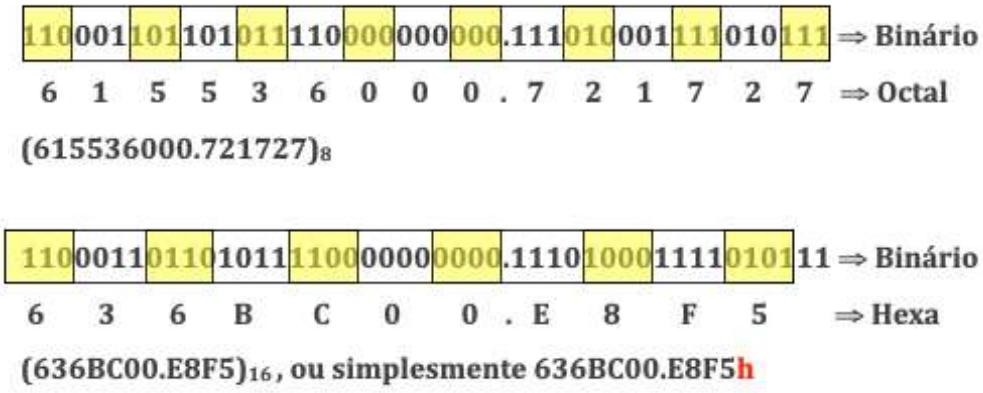
Feito isso, observa-se os restos das divisões de baixo para cima, e temos o número convertido!

Obviamente isto é apenas uma técnica que repete o método anterior, baseado na notação posicional. No entanto, ela pode parecer mais fácil em termos práticos.

$$62 = (76)_8$$

Após treinar com outros exemplos, você não terá dificuldades para converter números entre bases diferentes. Esta habilidade é importante para completar algumas atividades que você desenvolverá durante os seus estudos.

A conversão da base binária para as bases octal e hexadecimal, e vice-versa, pode ser feita de forma mais simples e direta. Para isto basta fazer a troca de dígitos octais por grupos de 3 dígitos binários, e de hexadecimais para grupos de 4 dígitos binários. A figura abaixo demonstra este processo. Note que, diante da ausência de dígitos à esquerda, ou à direita depois da vírgula, basta considerá-los com o valor zero, já que zeros à esquerda não tem valor algum, independente da base de numeração.



5.5 Conjuntos numéricos

Números binários podem representar qualquer número real (R). No entanto, a representação destes números pode ser diferente em função do seu tipo.

Números naturais (N) são representados diretamente das formas que vimos nos exemplos anteriores. Existem, no entanto, representações específicas para números inteiros negativos (Z), e para números racionais (Q) e irracionais (I).

Nas linguagens de programação, cabe ao programador escolher o tipo de representação mais adequada para o armazenamento de uma determinada informação numérica.

5.6 Números negativos

É importante lembrar que é possível representar números de qualquer tipo, como por exemplo números fracionários e negativos, em todas as bases de numeração.

Para representar números negativos, utiliza-se a notação conhecida como “Complemento de 2”. Neste sistema, o bit mais significativo identifica o sinal. O valor 1 torna o número negativo, enquanto o zero identifica um número positivo.

Para representar um número negativo, subtraímos 1 do valor positivo correspondente, e invertemos todos os bits. O processo contrário envolve inverter todos os bits, e somar o número 1.

Esta representação, largamente utilizada, elimina a possibilidade do "zero negativo", já que o valor zero tem apenas uma representação. Além disto, a representação permite a operação direta através da soma de parcelas positivas e negativas.

Vamos ver um exemplo somando os números -5 e 3 ?

-5: 0101 (5 em binário) → 0100 (subtraindo-se 1 do número) → **1011** (invertendo todos os bits)

2: 0010 (2 em binário)

Somando os dois números, temos

1011

0010 +

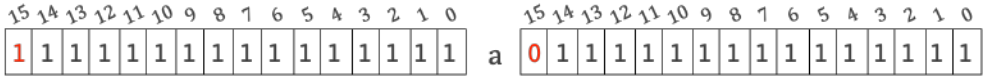
1101

1101 é negativo (primeiro dígito é 1). Qual o valor representado?

1101 → 0010 (invertendo todos os bits) → 0011 (somando 1 ao número) = -3 em decimal

Veja que a operação resultou correta.

É importante atentar para o fato de que o bit mais significativo fica reservado, sendo assim números escritos em complemento de 2 exigem 1 bit a mais do que números convencionais. Nas linguagens de programação, as variáveis que armazenam números em complemento de 2 são identificadas como ***signed***, enquanto as convencionais são identificadas como ***unsigned***. Abaixo vemos os limites negativo e positivo para o armazenamento de um número inteiro em uma variável *int signed* de 16 bits:



Em decimal, as quantidades podem variar de -32.768 a +32.767. Se não houvesse valor negativo, poderíamos armazenar inteiros entre 0 e 65.535 aproveitando todos os 16 bits.

5.7 Números fracionários e dízimas

Números fracionários são representados naturalmente através dos dígitos após a vírgula, seguindo a notação convencional conforme visto. Estes são os chamados números fracionários de ponto fixo. No entanto, é comum precisarmos utilizar números onde a vírgula pode se deslocar, de forma a permitir a representação de mais dígitos significativos.

Computadores utilizam a notação científica para isso. Qualquer número fracionário em binário é representado com duas partes: mantissa e expoente. Tanto a mantissa quanto o expoente podem ser positivos ou negativos, e no caso de negativos adotam a notação de complemento de 2, como já vimos. Neste caso, a precisão é determinada pela quantidade de bits para representação da mantissa, e a faixa de números que podem ser representados é determinada pelo número de bits do expoente. Novamente cabe ao programador escolher o formato desejado definindo o tipo de variável. Variáveis do tipo ***float***, por exemplo, oferecem precisão e faixa de valores menores que variáveis do tipo ***double***. Por outro lado, elas ocupam um espaço menor.

Quando usamos notação científica na base 10, as mantissas são normalizadas para conterem apenas uma parte fracionária, com o primeiro dígito depois da vírgula diferente de zero. No caso dos números binários, considerando que o primeiro bit representa o sinal, a mantissa deve começar com:

$(0,1)_2$ para números positivos;

$(1,0)_2$ para números negativos;

Números negativos, portanto, têm expoentes maiores que os mesmos números quando positivos.

A disposição dos bits no armazenamento de números de ponto flutuante pode ser feita seguindo diversos formatos, em função do fabricante ou família de processador ou microcontrolador.

Há, no entanto, uma proposta de padronização feita pelo IEEE (Instituto de Engenheiros Elétricos e Eletrônicos); trata-se da norma IEEE754. Nela, o bit mais significativo representa o sinal, seguido dos bits que representam o expoente. Os bits menos significativos são destinados à mantissa.

	Simples (32 bits)	Duplo (64 bits)	Quádruplo (128 bits)
Sinal (S)	1 bit	1 bit	1 bit
Expoente (E)	8 bits	11 bits	15 bits
Mantissa (M)	23 bits	52 bits	112 bits

Existem, no entanto, mais detalhes. Como representar o valor zero e infinito, por exemplo. Outra questão é como fazer operações básicas como soma, subtração, multiplicação e divisão.

Como os algoritmos envolvidos nas operações com ponto flutuante podem ser muito complexos, é comum existirem unidades aritméticas especializadas nestes cálculos. Aplicações que realizam cálculos intensivos, como jogos de ação, por exemplo, têm sua performance determinada pela capacidade e número de unidades de ponto flutuante.

Um caso curioso ocorre com as dízimas. Números que são dízimas em uma base podem não ser dízimas em outra, e vice-versa. Um exemplo clássico é o número $(0,1)_{10}$, que não pode ser representado de forma exata em binário com um número finito de dígitos.

$(0,1)_{10} = (0,000110011...)_2$

Este é o motivo pelo qual às vezes temos resultados "estranhos" nos programas de computador que desenvolvemos, ou mesmo ao usar algumas calculadoras. Para evitar este tipo de falha, recomenda-se usar aproximações para cálculos computacionais, preservando apenas as casas decimais essenciais.

6 Tabela ASCII

Os números também podem, e são utilizados, para representar outros símbolos como letras, caracteres especiais etc. Utilizam-se tabelas padronizadas que associam números específicos a cada um destes símbolos. A mais conhecida destas tabelas é a tabela ASCII. Você pode precisar consultar esta tabela para algumas operações no seu computador.

A tabela tem algumas curiosidades. Uma delas é que os códigos da representação de letras minúsculas são iguais aos códigos das maiúsculas somados ao código do espaço em branco.

Uma constatação importante é que operações que julgávamos ser orientadas a caracteres, como por exemplo a ordenação alfabética, na verdade são operações aritméticas!

Bin	Oct	Dec	Hex	Sinal
0010 0000	040	32	20	(espaço)
0010 0001	041	33	21	!
0010 0010	042	34	22	"
0010 0011	043	35	23	#
0010 0100	044	36	24	$
0010 0101	045	37	25	%
0010 0110	046	38	26	&
0010 0111	047	39	27	'
0010 1000	050	40	28	(
0010 1001	051	41	29	)
0010 1010	052	42	2A	*
0010 1011	053	43	2B	+
0010 1100	054	44	2C	,
0010 1101	055	45	2D	-
0010 1110	056	46	2E	.
0010 1111	057	47	2F	/
0011 0000	060	48	30	0
0011 0001	061	49	31	1
0011 0010	062	50	32	2
0011 0011	063	51	33	3
0011 0100	064	52	34	4
0011 0101	065	53	35	5
0011 0110	066	54	36	6
0011 0111	067	55	37	7
0011 1000	070	56	38	8
0011 1001	071	57	39	9
0011 1010	072	58	3A	:
0011 1011	073	59	3B	;
0011 1100	074	60	3C	<
0011 1101	075	61	3D	=
0011 1110	076	62	3E	>
0011 1111	077	63	3F	?

Bin	Oct	Dec	Hex	Sinal
0100 0000	100	64	40	@
0100 0001	101	65	41	A
0100 0010	102	66	42	B
0100 0011	103	67	43	C
0100 0100	104	68	44	D
0100 0101	105	69	45	E
0100 0110	106	70	46	F
0100 0111	107	71	47	G
0100 1000	110	72	48	H
0100 1001	111	73	49	I
0100 1010	112	74	4A	J
0100 1011	113	75	4B	K
0100 1100	114	76	4C	L
0100 1101	115	77	4D	M
0100 1110	116	78	4E	N
0100 1111	117	79	4F	O
0101 0000	120	80	50	P
0101 0001	121	81	51	Q
0101 0010	122	82	52	R
0101 0011	123	83	53	S
0101 0100	124	84	54	T
0101 0101	125	85	55	U
0101 0110	126	86	56	V
0101 0111	127	87	57	W
0101 1000	130	88	58	X
0101 1001	131	89	59	Y
0101 1010	132	90	5A	Z
0101 1011	133	91	5B	[
0101 1100	134	92	5C	\
0101 1101	135	93	5D	]
0101 1110	136	94	5E	^
0101 1111	137	95	5F	_

Bin	Oct	Dec	Hex	Sinal
0110 0000	140	96	60	`
0110 0001	141	97	61	a
0110 0010	142	98	62	b
0110 0011	143	99	63	c
0110 0100	144	100	64	d
0110 0101	145	101	65	e
0110 0110	146	102	66	f
0110 0111	147	103	67	g
0110 1000	150	104	68	h
0110 1001	151	105	69	i
0110 1010	152	106	6A	j
0110 1011	153	107	6B	k
0110 1100	154	108	6C	l
0110 1101	155	109	6D	m
0110 1110	156	110	6E	n
0110 1111	157	111	6F	o
0111 0000	160	112	70	p
0111 0001	161	113	71	q
0111 0010	162	114	72	r
0111 0011	163	115	73	s
0111 0100	164	116	74	t
0111 0101	165	117	75	u
0111 0110	166	118	76	v
0111 0111	167	119	77	w
0111 1000	170	120	78	x
0111 1001	171	121	79	y
0111 1010	172	122	7A	z
0111 1011	173	123	7B	{
0111 1100	174	124	7C	|
0111 1101	175	125	7D	}
0111 1110	176	126	7E	~

7 Computador – uma máquina de informações

Para capturar, armazenar, processar e fornecer informações, tanto como matéria-prima quanto como produto final, a "máquina" (dispositivo computacional) precisa trabalhar com representações destas informações, que, como vimos, estão codificadas em formato digital e binário.

Nas primeiras tentativas de colocar uma máquina para trabalhar com informações, o homem utilizou dispositivos mecânicos. Na época, as aplicações para a eletricidade eram desconhecidas, e apenas as engrenagens e componentes mecânicos estavam disponíveis. Obviamente, a complexidade envolvida na construção destas máquinas era demasiadamente elevada, o que as tornavam caras e raras, e na maior parte das vezes, pouco funcionais. Alguns exemplos, no entanto, foram surpreendentes.

7.1 O ábaco

Trata-se, em termos simples, de um mecanismo que permite estender o uso dos dedos para contagens e cálculos. Por este motivo, alguns não incluem este artefato como parte da história dos computadores modernos, embora o mesmo seja fundamental para ilustrar o surgimento da representação posicional que estudamos na unidade anterior.

O ábaco provavelmente surgiu na Mesopotâmia (na região onde hoje ficam o Iraque, o Afeganistão e a Síria) há mais de 5.000 A.C. Ao longo de sua história o mesmo foi adotado por diversas civilizações, em formatos diferentes, mas sempre baseado no mesmo princípio de utilização. Ele é utilizado até hoje para ensinar operações aritméticas a crianças em diversos países, como por exemplo na China.

O ábaco mais comum hoje utiliza a base 10, mas algumas versões utilizadas na antiguidade já foram baseadas no sistema de numeração sexagesimal (utilizado até hoje na representação dos minutos e segundos).

7.2 A máquina de Anticítera

Descoberta na costa da ilha grega de Anticítera em 1901, junto a estátuas e outros objetos antigos, inicialmente o artefato não chamou muito a atenção da comunidade científica. Posteriormente, depois de ser estudada com maior profundidade a partir de 1950, descobriu-se que se tratava de um computador analógico! Datada do ano de 87 A.C., a máquina de Anticítera era capaz de prever eventos astronômicos com razoável precisão.

Construída com 27 engrenagens em bronze, ele revelou uma habilidade construtiva dos gregos que não conhecida, pois se tratava de um mecanismo muito sofisticado para a época. Entre outras inovações, a máquina utilizava, por exemplo, engrenagens diferenciais, as quais até então se acreditava terem sido inventadas apenas no século XVI.

7.3 A Máquina Analítica

Projetada pelo mesmo cientista e matemático, a máquina analítica era sucessora da "Máquina Diferencial", e foi apresentada em 1837. Charles Babbage, considerado hoje como o "pai da computação" desenvolveu ambos os projetos, que infelizmente não chegaram a ser construídos por problemas de financiamento. No entanto, o projeto antecipava em cerca de um século as características básicas dos primeiros dispositivos computacionais, características estas que até hoje são adotadas nos computadores modernos.

Um dos princípios inovadores na época era a capacidade de "programação". Programas e dados, representados na base 10, eram transferidos e recuperados da máquina através de cartões perfurados.

A máquina também tinha uma memória capaz de armazenar até 1000 números de 40 dígitos (cerca de 16 KiloBytes), uma Unidade Lógica e Aritmética, e uma Unidade Centra de Processamento.

Uma descrição de sua operação em inglês foi publicada em 1842. Esta publicação foi bastante comentada por Ada Lovelace, que a utilizou para desenvolver um método de cálculo de números de Bernoulli (na verdade, um **algoritmo**). Por este trabalho, baseado na Máquina Analítica de Charles Babbage, Ada é considerada como a primeira programadora de computadores da história.

7.4 A Máquina de Turing

Desenvolvida em 1940, durante o esforço de guerra dos aliados na 2ª Guerra Mundial, a "Bombe" foi um dispositivo fundamental para a decodificação das mensagens criptografadas do exército alemão. A invenção de Alan Turing é considerada por muitos como um dos principais motivos da vitória dos aliados. Recentemente, o filme "O Jogo da Imitação", ganhador do Oscar em 2015, contou os detalhes sobre a criação e uso do dispositivo.

Muito embora tenha sido utilizada com sucesso, a Máquina de Turing foi construída com uma tecnologia eletromecânica, tal como outros dispositivos da época, a exemplo do Mark I. Esta tecnologia dificultava sua ampliação e aumento de performance.

De qualquer forma, a arquitetura e o projeto conceitual da Máquina de Turing eram inigualáveis. Ela acabou estabelecendo a base teórica de todos os dispositivos computacionais, e até hoje influencia o projeto de novos processadores e computadores. Isso tudo além de sua importância histórica.

7.5 ENIAC

A máquina de Turing acabou inspirando o desenvolvimento de computadores de grande escala, porém com base em componentes melhores e mais adequados do que os dispositivos eletromecânicos do projeto original.

Um dos computadores de maior destaque foi o ENIAC (*Electronic Numerical Integrator and Computer*), que entrou em operação em fevereiro de 1946, ou seja logo após o término da 2ª Guerra Mundial.

Embora tenha sido concluído após a guerra, seu desenvolvimento também foi parte do esforço de guerra. Ele foi criado primariamente para cálculo de tabelas de artilharia, que demoravam muito quando eram realizados por "computadores" humanos (assim eram chamadas as mulheres especializadas em cálculos sofisticados, na época).

Um dos maiores problemas do ENIAC era a sua programação, que era feita via hardware, através de interruptores e conexão de cabos. Cada novo programa exigia horas de trabalho para reconfiguração dos interruptores. Já o seu "sistema operacional" era armazenado em cartões perfurados.

7.6 O IAS, ou Máquina de von Neumann

Na década seguinte ao projeto de Turing, o matemático John von Neumann, do Instituto de Estudos Avançados de Princeton (a sigla IAS vem do nome do instituto) também se debruçou na criação de um dispositivo computacional com base nos conceitos de Turing.

Com o armazenamento dos programas na memória do computador, o IAS revolucionou os computadores, que se tornaram mais rápidos e práticos. Além disto, apesar de ter uma implementação fiel ao projeto do Alan Turing, o IAS incorporou na prática diversos avanços, e é utilizado até hoje como modelo de arquitetura de computadores.

A chamada "Máquina de von Neumann" é um modelo teórico que até hoje serve como base para o projeto dos computadores modernos. Estudaremos melhor as características do projeto na próxima unidade, quando conheceremos o IAS em detalhes.

7.7 A troca dos relês pelas válvulas

Os computadores citados nos exemplos anteriores evoluíram de dispositivos mecânicos para eletromecânicos, que utilizavam relés como interruptores acionados eletricamente.

Embora funcionais, os relés tinham limitações de performance, já que se tratava de componentes eletromecânicos. A primeira evolução foi a substituição dos mesmos pelas válvulas eletrônicas, que embora não trouxessem redução significativa de tamanho nem de confiabilidade, eram muito mais rápidas que os relês.

7.8 A troca das válvulas pelos transistores

Um dos maiores problemas dos primeiros computadores, mesmo baseados em válvulas, era a baixa confiabilidade. Como os dispositivos utilizavam milhares de válvulas, não era raro encontramos problemas como falhas nos contatos das válvulas, ou mesmo a sua queima.

É desta época inclusive o surgimento do termo "*bug*" (besouro, em inglês), que surgiu devido a este problema. A falta de contato em uma das válvulas de um grande computador havia sido provocada pela presença de um besouro no soquete de uma das válvulas, provocando erros operacionais. A descoberta da origem do problema acabou associando ao inseto os erros nos programas de computador.

Além disto, as válvulas consumiam muita energia, já que funcionavam como lâmpadas incandescentes, onde um filamento precisava estar aceso para que a válvula funcionasse adequadamente. Outro problema era o espaço ocupado pelas válvulas; como os computadores utilizavam milhares delas, os computadores eram máquinas imensas e muito pesadas.

Estes problemas puderam ser resolvidos pela substituição das válvulas pelos transistores, recentemente inventados durante a década de 50. Além de pequenos, com pequeno consumo de energia, os transistores continuavam sendo muito rápidos tal como as válvulas, e ainda dispensavam as altas tensões de operação necessárias no caso das válvulas.

7.9 O surgimento dos circuitos integrados

A junção de cada vez mais transistores em um único invólucro, chegando até a milhões destes dispositivos nos processadores mais modernos, aumentou a confiabilidade, reduziu o consumo de energia e diminuiu muito o tamanho dos dispositivos computacionais. Estes conjuntos foram chamados de "circuitos integrados", ou mais popularmente como "chips".

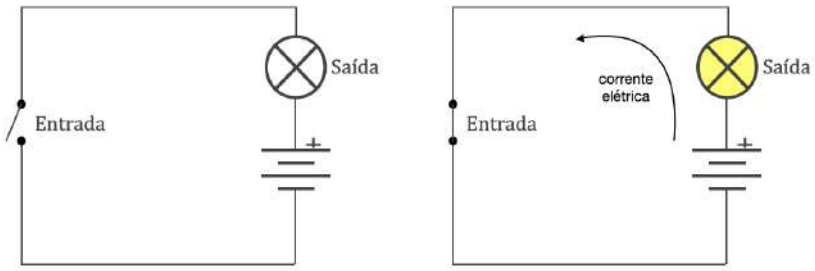
Os conjuntos inicialmente formavam blocos funcionais conhecidos como "portas lógicas". A junção de diversas portas lógicas permitiu a criação de blocos de função específica, como somadores, registradores de deslocamento, memórias etc. Hoje em dia temos processadores completos, inclusive com pares dos chips periféricos, todos agrupados em uma única "pastilha" semicondutora de dimensões reduzidas, como por exemplo nos SoC (*System-on-a-Chip*).

Conhecer e entender estes conjuntos e dispositivos é um dos principais objetivos deste curso. Veremos isto ao longo dos próximos itens deste documento.

8 Circuitos elétricos

Antes de vermos a implementação dos blocos básicos de um dispositivo computacional, precisamos rever o princípio de operação de qualquer circuito elétrico, o que você já deve ter visto ao longo das aulas de física no segundo grau.

Em um circuito elétrico, uma fonte de tensão elétrica (uma pilha, por exemplo) provoca a circulação física de elétrons (ou lacunas no sentido matemático) através de condutores. No caminho entre os dois polos desta fonte, os elétrons podem ser forçados a passar por componentes que transformarão a energia acumulada como diferença de potencial da fonte em outro tipo de energia (térmica, luminosa, eletromagnética etc.). Veja o exemplo a seguir:



No exemplo, podemos considerar a chave interruptora à esquerda de ambas as figuras como uma "entrada". Ela assume o valor lógico 0 (falso) quando está aberta, e o valor 1 (verdadeiro) quando está fechada. Uma chave aberta impede a circulação de corrente elétrica, mantendo a saída (lâmpada ao lado direito de ambas as figuras) apagada, ou seja, com valor 0 (falso). Já a chave fechada permite a circulação da corrente, como vemos na figura do lado direito, acendendo a lâmpada, que assume o valor verdadeiro (1).

Além de acender uma lâmpada, a energia elétrica fornecida pela bateria pode ser usada para diversas aplicações, como o aquecimento de um condutor específico, geração de campos magnéticos (em um eletroímã, ou relé, por exemplo), trabalhos mecânicos (pelo acionamento de motores), entre outros. Nos exemplos que estudaremos e veremos no simulador de circuitos lógicos, utilizaremos uma fonte de alimentação externa que alimentará diversos dispositivos de entrada e saída, além dos próprios componentes responsáveis pela "lógica" do dispositivo computacional.

É trabalho do projetista do circuito lógico calcular as intensidades aplicáveis de tensão e corrente no circuito para garantir uma operação satisfatória e segura, além de conectar corretamente os componentes do circuito. Nos exemplos práticos que veremos no simulador de circuitos lógicos, os "cálculos" serão feitos pelo professor, mas as conexões precisarão ser realizadas cuidadosamente para garantir o sucesso dos experimentos.

Quanto aos cálculos, caso você precise fazer você mesmo, basta lembrar, por exemplo, da lei de ohm que você estudou no segundo grau. Lembra? Na prática, a aplicação de uma diferença de potencial de tensão (V) pela fonte provoca a circulação de uma corrente elétrica (i) que depende da resistência elétrica (R) que os condutores e componentes oferecerão, ao longo do circuito, à passagem da corrente ($i = V / R$). Em algumas situações específicas, talvez você precise de um pouco mais de conhecimento, mas você já pode começar mesmo apenas com estes conhecimentos básicos – circuitos digitais não são complicados!

Revisando, é necessário que exista um "circuito fechado", ou seja, um caminho sem interrupções entre os dois polos da fonte de tensão, para que ele funcione. A interrupção do circuito, que no nosso exemplo foi realizada através de um interruptor físico, também pode ser feita por interruptores acionados eletricamente (relés), válvulas, ou semicondutores. Não por coincidência, estes foram e são os componentes típicos em um dispositivo computacional.

9 Funções Lógicas Binárias em circuitos elétricos

Através da implementação de arranjos elementares de interruptores (utilizados como "entradas"), e de LEDs luminosos (utilizados como "saídas"), é perfeitamente possível sintetizar algumas funções lógicas básicas (utilizamos LEDs substituindo as lâmpadas para maior simplicidade e menor consumo de energia). Estas funções são similares às funções dos blocos básicos de um dispositivo computacional, muito embora sejam implementadas de forma mais confiável e eficiente nestes últimos.

Antes de mais nada, precisamos associar os algarismos binários às suas representações. Nos circuitos elétricos que citaremos, o algarismo 0 (Zero) é tipicamente associado ...

... na entrada, a um circuito aberto (interrompido)

... na saída, a um LED Apagado

No caso do algarismo 1 (Um), a associação é ...

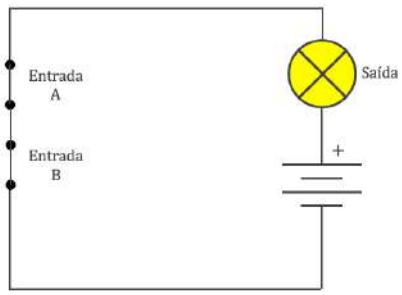
... na entrada, a um circuito fechado

... na saída, a um LED Aceso

9.1 Lógica E (AND)

A lógica AND estabelece uma saída verdadeira apenas quando TODAS as entradas forem verdadeiras. A existência de apenas uma entrada falsa torna a saída falsa. Uma analogia simples seria colocar duas condições (entradas) para que você pudesse ir à praia (saída): o dia estar ensolarado (entrada 1), e você ter a grana para curtir a praia (entrada 2). Na prática, você só iria à praia (saída verdadeira) se as duas entradas forem verdadeiras (fazer sol, e ter grana). Se qualquer uma das duas entradas for falsa, na prática a saída também o será.

Estendendo o exemplo, é claro que é possível implementar a lógica E com duas ou mais entradas, a depender do circuito lógico que está sendo implementado. No entanto, se montarmos um circuito elétrico para representar esta lógica, teríamos algo assim, ainda usando interruptores e lâmpadas:



Lógica "E" (AND)

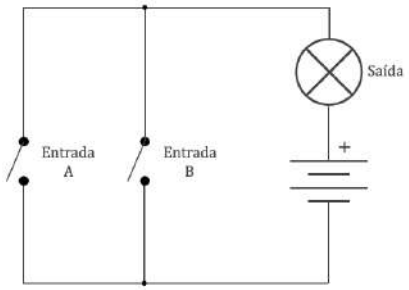
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Observe que a lâmpada está acesa apenas porque os dois interruptores estão fechados. A tabela do lado direito representa todas as opções de comportamento em função do estado das duas entradas (falsas ou verdadeiras). Esta tabela é conhecida como **tabela verdade**, e é bastante utilizada na análise do comportamento de circuitos lógicos.

9.2 Lógica OU (OR)

A lógica OR estabelece uma saída falsa apenas quando TODAS as entradas forem falsas. A existência de uma única entrada verdadeira torna a saída verdadeira. Assim como no caso do E, também é possível implementar a lógica OR com duas ou mais entradas, a depender do circuito que está sendo implementado.

Analogamente, o circuito elétrico que representa esta lógica fica assim:



Lógica "OU" (OR)

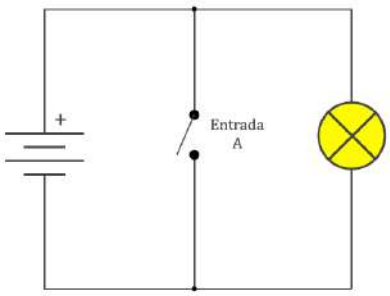
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Observe que a lâmpada está apagada apenas porque os dois interruptores estão abertos. A tabela verdade do lado direito representa o comportamento para este caso.

9.3 Lógica NÃO (NOT)

A lógica NOT estabelece uma saída com valor oposto à entrada, ou seja, se a entrada for verdadeira, a saída será falsa, e vice-versa. Obviamente, a lógica NOT só admite uma única entrada para uma saída.

O circuito ficaria assim:



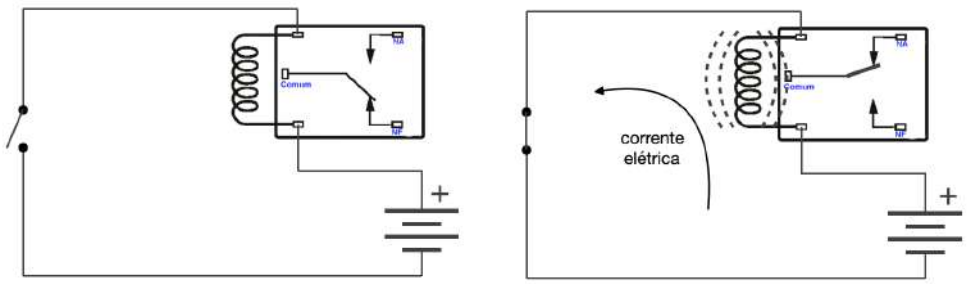
A	S
0	1
1	0

Observe que a lâmpada está acesa apenas porque o interruptor está aberto. A tabela verdade do lado direito representa o comportamento para este caso. No entanto, temos um problema. Este circuito não é funcional!

Para apagar a lâmpada, somos forçados a provocar um curto-circuito, o que certamente provocará danos à bateria, ao interruptor, aos condutores, e a depender das potências envolvidas, pode ser inclusive perigoso!

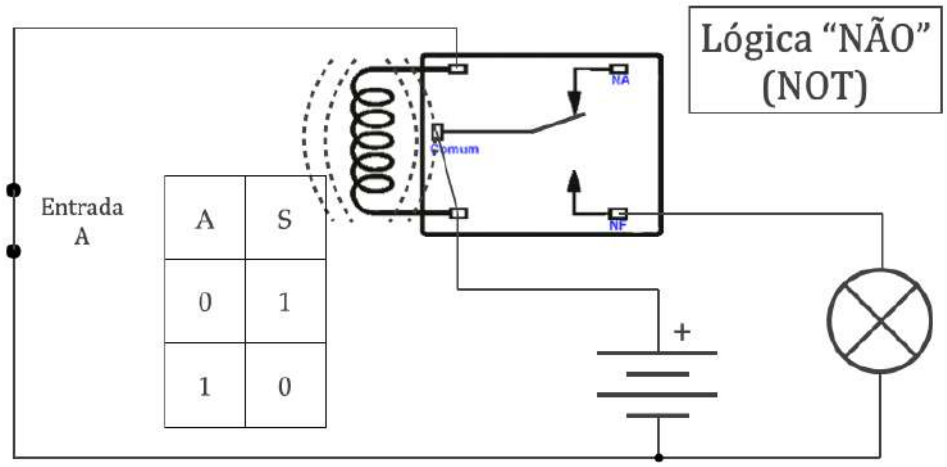
Diante disto, vamos precisar de um novo componente, que não só resolve este problema, como também permite a interconexão dos circuitos elétricos. A interconexão é fundamental, já que é comum precisarmos de circuitos compostos de diversos circuitos lógicos concatenados.

Para isso os interruptores e lâmpadas intermediários foram substituídos por relês eletromecânicos. Um relê basicamente é uma chave eletromagnética, ou seja, uma chave que pode ser ligada através de um campo magnético, sem intervenção física/manual do usuário.



Na figura acima, vemos o relê nos dois estados: desligado à esquerda, e ligado à direita. Observe que a chave interna muda de posição quando é energizada.

Usando um relê, conseguimos montar um circuito elétrico funcional, conforme podemos ver na figura a seguir. Neste caso não há curto-circuito, nem risco, e a tabela verdade pode ser implementada.



Neste circuito acima, a lógica NÃO foi implementada sem riscos de curto-circuito ou dano nos componentes. Neste circuito você pode encostar!

10 Evolução da eletrônica nas "Gerações" de Computadores

Uma vez conhecidos estes três blocos básicos, a pergunta típica é: seria possível construir um computador utilizando apenas estes exemplos de circuitos?

A verdade é que, com apenas estes 3 blocos, teoricamente é possível construir um computador. Isto foi o que efetivamente foi feito no século passado, com a interligação de milhares destes blocos, ocupando grandes espaços, e com o consumo de muita energia elétrica (um relê moderno consome algo em torno de 3W, imagine um equipamento com milhares destes componentes).

Além disto, relés eletromecânicos ocupam muito espaço, são lentos e pouco confiáveis, ainda mais depois de muito tempo de uso. Mesmo assim, como já vimos, foram estes componentes que foram usados nos primeiros computadores não-mecânicos.

Como já vimos, estes problemas foram tratados. Chegamos à chamada **1ª Geração dos Computadores**, com a substituição dos relês mecânicos por válvulas a vácuo. A válvula é um componente eletrônico muito utilizado no passado (ainda existe uma dentro do seu micro-ondas, chamada de "Magnetron"). Entre as principais aplicações da válvula está a de "relê eletrônico", mais confiável e muito mais rápido. Isso permitiu a criação de computadores melhores.

Na 2ª Geração, as válvulas foram substituídas pelos transistores, o que reduziu o seu tamanho e consumo de energia. Na 3ª geração, vieram os chamados "circuitos integrados", avós dos *chips* atuais, que ocupavam ainda menos espaço, e consumiam menos energia.

No início, estes Cis (Circuitos Integrados) abrigavam conjuntos de circuitos eletrônicos que simulavam, usando componentes mais modernos, os mesmos circuitos lógicos básicos que acabamos de ver.

11 Portas Lógicas Básicas

Conforme vimos anteriormente, os blocos elementares de um dispositivo computacional representam funções lógicas básicas. Com a evolução da eletrônica, estes circuitos passaram a ser representados por "Portas Lógicas". Cada porta lógica, que tem representação gráfica específica, possui um conjunto fixo de entradas (tipicamente 2) e tipicamente apenas uma saída.

11.1 Porta AND (E)

Implementa a função lógica E. Tipicamente possui duas, mas pode ter mais entradas. O símbolo à direita é utilizado para identificá-la nos circuitos digitais, inclusive nas folhas de dados com as especificações dos Cis.

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

11.2 Porta OR (OU)

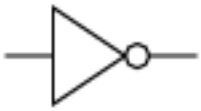
Implementa a função lógica OU. Assim como a porta E, tipicamente tem duas, mas pode ter mais entradas. O símbolo também está à direita.

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

11.3 Porta NOT (NÃO)

Implementa a função NÃO. A porta NOT tem apenas uma entrada. Pode vir acoplada à saída das portas lógicas anteriores, que passam a se chamar de NAND e NOR.

A	S
0	1
1	0



11.4 Porta XOR (OU EXCLU

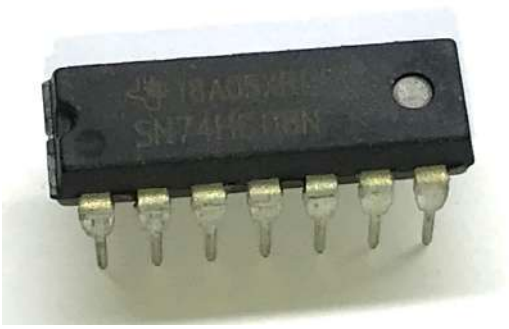
Implementa o circuito lógico OU EXCLUSIVO, que embora possa ser obtido através da conexão das portas lógicas básicas vistas anteriormente, foi implementado com uma nomenclatura e símbolo específico. É tipicamente utilizado para comparar dois valores lógicos, já que sua saída assume o valor zero (falso) quando as entradas são iguais, e o valor um (verdadeiro) quando as entradas são diferentes. Uma das aplicações mais conhecidas é nos circuitos aritméticos binários, que veremos em breve.

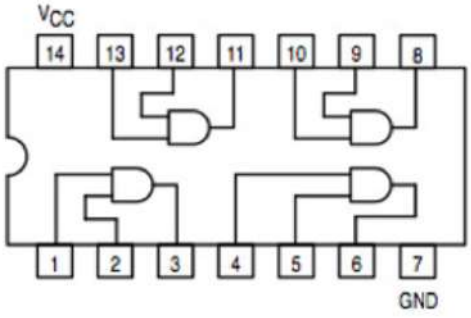
A porta XOR por definição possui sempre duas entradas. Quando à saída da mesma é conectada a uma porta NOT, ela se transforma em uma porta XNOR. O símbolo está à direita.

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

As portas lógicas são encapsuladas dentro de Cis, que normalmente oferecem diversas portas em um único "chip".

Na foto a seguir vemos o CI 74HC08, que contém quatro portas lógicas AND. Do lado direto, vemos as conexões internas do CI para uso em circuitos digitais. São 14 pinos, numerados de 1 a 14 no sentido anti-horário, começando do pino abaixo do chanfro do lado esquerdo. Note que, além das 4 portas E, que ocupam 12 dos 14 pinos do componente, temos dois pinos adicionais para a alimentação do CI. O pino 14 (Vcc) deve ser conectado ao polo positivo da alimentação (5 volts), e o pino 7 (GND, que é a abreviatura de GrouND, ou terra) deve ser ligado ao polo negativo (0 volts).





Os valores lógicos verdadeiro (1) e falso (0) são representados por tensões elétricas. Quando a entrada de uma das portas lógicas é conectada a uma tensão igual ou

próxima a Vcc (5V), ela será considerada verdadeira. Se conectarmos a mesma a uma tensão igual ou próxima a GND (0V), ela será considerada falsa. As saídas das portas (pinos 3, 6, 8 e 11) apresentarão tensões elétricas similares para representar os valores lógicos verdadeiro e falso de suas respectivas portas.

Nos experimentos que você pode fazer no simulador de circuitos lógicos, além do 74HC08, você pode utilizar outros Cis, como por exemplo:

CI 74HC00 - Quatro Portas Lógicas NAND

CI 74HC02 - Quatro Portas Lógicas NOR

CI 74HC04 - Seis Portas Lógicas NOT

CI 74HC32 - Quatro Portas Lógicas OR

CI 74HC86 - Quatro Portas Lógicas XOR

Para verificar quais as funções de cada pino dos Cis acima, você pode pesquisar na Internet, e guardar contigo as características destes componentes. Isso pode lhe ajudar na preparação para os experimentos no simulador de circuitos lógicos.

12 Um exemplo de circuito Aritmético

Tal como vimos, as portas lógicas básicas podem ser concatenadas para formar os componentes de um computador típico. Para começar a entender como isso pode ser feito, vamos analisar uma operação aritmética básica, a soma.

No exemplo abaixo, temos a soma de dois números binários de oito bits:

```
    1   1 1 1
1 0 0 1 0 1 1 1
0 1 0 1 0 1 0 1  +
-----------------
1 1 1 0 1 1 0 0
```

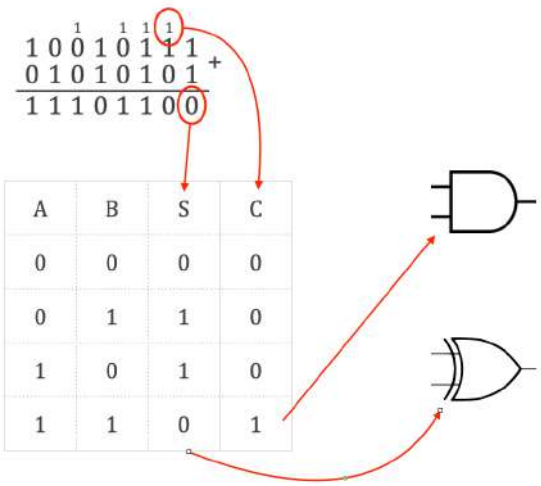
Vamos acompanhar o processo de soma. Ela é feita da direita para a esquerda, começando do dígito menos significativo. No primeiro, ao somar um + um, obtivemos “1 0”, ou seja, “0, e vai 1”. Lembre-se que no sistema de numeração binário não existe o número “2”, logo precisamos representar a quantidade dois com dois dígitos, ou seja, “10”.

A partir do segundo dígito, precisamos somar três dígitos, já que temos o “vai um” do dígito anterior. Ou seja, nos oito bits que foram somados, o único dígito onde somamos apenas dois bits é o menos significativo, já que ele não tem dígito anterior.

Sendo assim, temos dois tipos de operação na soma de dois números binários: aquelas em que somamos apenas dois bits, e aquelas em que somamos três bits. Em ambas, temos duas saídas: a soma propriamente dita, e o dígito de “vai um”, que será a uma das entradas na soma do próximo dígito.

Para executar os dois tipos de operação, temos dois tipos de circuito: um somador com duas entradas, também chamado de "meio somador", e um somador com 3 entradas, que é chamado de "somador completo".

Sempre que formos somar "n" bits, precisaremos de um meio somador, e de "n-1" somadores completos.

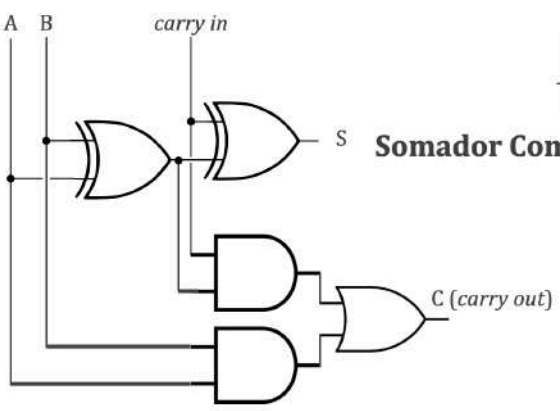


A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

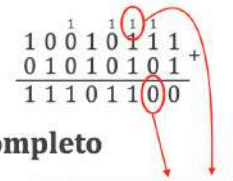
Como qualquer operação lógica, podemos obter uma tabela verdade que representa os valores das saídas com base nas entradas. Veja ao lado a tabela verdade do meio somador.

Observando os valores das saídas S (soma) e C (*carry*, ou "vai-um"), percebe-se claramente que são equivalentes às tabelas verdade respectivamente da porta OU EXCLUSIVO e da porta E.

Veremos em breve como estas equivalências são representadas em um circuito lógico utilizando as respectivas portas conectadas às entradas, ou seja, aos bits que serão somados.



Somador Completo



A	B	Ci	S	Co
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

Para completar, vamos incluir também o somador completo. Neste caso, temos um circuito um pouco mais complexo. para executar a lógica do somador completo. Embora continue utilizando portas OU EXCLUSIVO e E, ele exige uma quantidade maior de portas para executar o cálculo.

Como vimos, neste caso temos três entradas (além dos dois bits a serem somados, temos o "vai um" do dígito anterior), e duas saídas.

Podemos usar este exemplo para avaliar a complexidade de circuitos mais elaborados como o circuito de um computador. Considerando que o meio somador utiliza duas portas lógicas, e o somador completo cinco portas, para somar os dois Bytes (oito bits) do exemplo anterior, seriam necessárias quantas portas lógicas? E em um computador de 32 bits? Esta é uma boa explicação para a quantidade

gigantesca de portas lógicas que são necessárias para construir um computador moderno.

Respondendo à pergunta acima, para somar oito bits seriam necessárias 2 + 5 x 7 portas, ou seja, 37 portas lógicas. Para um computador de 32 bits, seriam necessárias 2 + 31 x 5 portas, ou seja, 157 portas!

Os circuitos aritméticos, como os somadores, e também outros circuitos especializados em outras operações matemáticas e lógicas são agrupados, em um computador moderno, na chamada ULA – Unidade Lógica e Aritmética.

Mas e na prática? Como conectamos os Cis para que eles realizem estas operações aritméticas? Para explicar melhor este processo, você pode assistir a dois vídeos que preparei sobre o tema.

No primeiro vídeo, veja a teoria por detrás da montagem de um circuito somador de dois números de dois bits:

http://y2u.be/erX7pJprlWI

No segundo vídeo, veja a montagem, na prática, deste circuito lógico, inclusive mostrando o resultado no formato decimal para o usuário:

http://y2u.be/w-yc9W6a0Vs

Assista aos vídeos! Eles podem ajudá-lo a fixar os conceitos vistos até agora.

13 Síntese de Circuito Lógico a partir da Tabela-Verdade

Ainda usando o exemplo anterior, vimos que era relativamente fácil perceber, no caso do circuito Meio Somador, qual era a sua tabela-verdade. No entanto, ainda no mesmo exemplo, o circuito do Somador Completo não era tão simples.

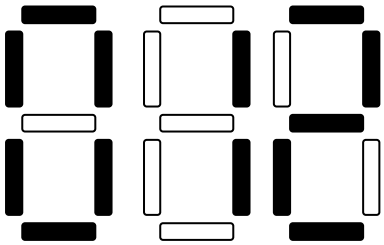
Dá para imaginar que existe algum método para obter, de forma sistematizada, o circuito lógico correspondente a uma determinada tabela-verdade, ou seja, deve ser possível "sintetizar" um circuito que realize a operação que desejamos.

A síntese de circuitos é útil não só para projetar circuitos lógicos que atendam a determinada condição – o que provavelmente não fará parte de sua atividade profissional futura - mas também para determinar como implementar uma "estrutura de decisão" em um determinado *software* que você está desenvolvendo. Isso vai garantir que ele realize – de forma simples - exatamente aquilo que você deseja com base nas informações recebidas. Parece interessante, não? Vamos ver como se faz.

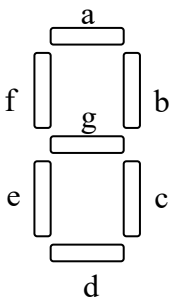
Na nossa 2ª Lista de exercícios, tivemos um exemplo básico de síntese de circuitos lógicos. Na 3ª questão da lista, tínhamos uma tabela verdade onde todas as saídas eram falsas, e apenas uma era verdadeira. Na execução do exercício, associamos este comportamento à porta AND, e com isso conseguimos resolver o problema. Já

na 4ª questão, a tabela verdade tinha apenas uma saída falsa, e todas as outras verdadeiras. Naquele caso, associamos o comportamento à porta OR.

No entanto, nem sempre uma tabela verdade possui apenas uma situação verdadeira, que nos permite associar o comportamento a uma porta AND; ou apenas uma situação falsa, o que nos permitiria usar uma porta OR. Muitas vezes as saídas assumem valores diversos, com mais de uma saída para cada uma das possibilidades apresentadas na tabela. Vejamos um exemplo:



O desenho ao lado representa o chamado "Display de 7 segmentos", muito utilizado para representar números em mostradores diversos, de relógios digitais a painéis de chamada para sistemas de atendimento. Você deve ter visto um deles, inclusive, no segundo vídeo do item anterior.



Neste tipo de display, temos 7 LEDs, identificados com as letras de "a" a "g", que acendem de acordo com o número binário que é entregue ao decodificador.

No exemplo ao lado, temos as representações dos números 0, 1 e 2 usando os sete segmentos. Se escolhermos o segmento "a", veremos que o mesmo acende para os seguintes valores: 0, 2, 3, 5, 6, 7, 8 e 9. O mesmo segmento fica apagado para os valores 1 e 4. Como sintetizar o circuito lógico que ativaria este segmento?

b3	b2	b1	b0	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Na tabela verdade ao lado, que representa o comportamento do segmento "a", temos oito saídas verdadeiras, e duas falsas. Existem também seis "saídas irrelevantes", que se referem às combinações de entradas que jamais ocorrem, pois são entradas correspondentes aos números maiores que 9 (neste caso, hexadecimais). Como não interessa qual a saída para combinações de entrada que nunca ocorrerão, tanto faz o valor da saída (0 ou 1), e por isso marcamos as mesmas com "x".

Existem duas possíveis soluções. Na primeira, montaríamos um circuito AND para cada uma das oito combinações em que o segmento se acende; na outra, um circuito OR para cada uma das duas combinações em que o segmento se apaga. Pelo grau de complexidade, precisamos escolher a segunda opção, onde teríamos apenas duas portas OR (na outra opção, teríamos que utilizar oito portas AND). Neste caso, é claro, vamos considerar que todas as saídas irrelevantes são iguais a 1.

Cada uma das portas OR representa, então, uma das situações em que a saída teria o valor 0. Lembrando da Lista de Exercícios, seria fácil montar o circuito lógico para garantir a saída falsa em cada uma das situações usando uma porta OR com quatro entradas. O problema, no entanto, é que precisamos juntar estes dois resultados, de forma a ter apenas uma saída para controlar o segmento.

Para fazer isto, usamos uma porta AND interligando as duas saídas das portas OR. Isso resolve o problema, já que ele manterá o segmento aceso apenas quando as duas saídas foram verdadeiras, e como elas são verdadeiras em todos os casos, exceto naqueles em que o segmento deve ficar apagado, o arranjo funciona perfeitamente.

Obtido o circuito lógico que acenderá corretamente o segmento "a" do display, precisamos repetir o procedimento para cada um dos outros segmentos. Teremos, portanto, sete circuitos, um para cada segmento do display.

Na tabela à direita, temos o comportamento do segmento "e". Ao contrário do segmento "a", neste caso vemos que ele acende menos frequentemente do que apaga, ou melhor, ele acende para os números 0, 2, 6 e 8, mas permanece apagado para os valores 1, 3, 4, 5, 7 e 9. Neste caso, também pela simplicidade, precisamos escolher a solução com portas AND, e não OR.

De forma oposta, devemos interligar as quatro portas AND por uma porta OR de quatro entradas para controlar o segmento.

b3	b2	b1	b0	e
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Se você se dedicar a fazer o mesmo trabalho para todos os segmentos, vai perceber que a complexidade do circuito completo é significativa. Isso nos leva a uma pergunta: seria possível "simplificar" os circuitos?

14 Simplificação de Circuitos Lógicos

Em termos práticos, simplificar um circuito significa ocupar menos espaço, gastar menos energia, e também aumentar a confiabilidade, pois quanto menos componentes, menor a probabilidade de defeitos.

Na verdade, nós até já falamos de uma técnica para simplificar os circuitos. A consideração de "saídas irrelevantes", por exemplo, na prática reduz a complexidade dos circuitos.

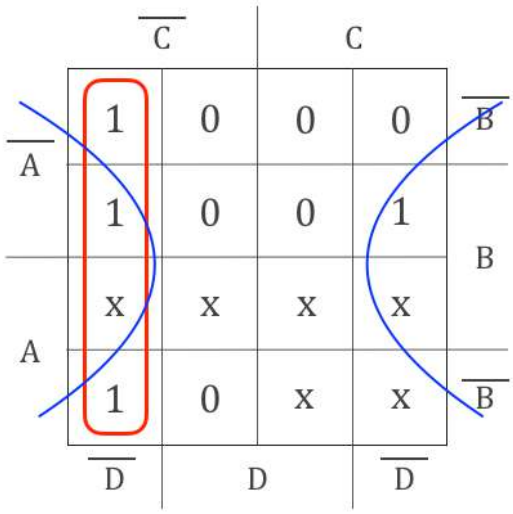
Outra técnica bastante simples é reaproveitar partes de expressões lógicas que porventura já existam para outras saídas. Neste caso, como temos sete diferentes saídas, certamente conseguiremos fazer isto, e acabamos também por reduzir a complexidade do circuito final.

A prática mais adequada, no entanto, é a utilização de técnicas de simplificação das expressões lógicas. Existem diferentes métodos para realizar isto.

Um deles é a simplificação direta da expressão usando a chamada "Álgebra de Boole". A álgebra de Boole tem diversas aplicações, e não apenas em Arquitetura de Computadores. No entanto, o estudo de suas operações, propriedades e técnicas está fora do escopo de nossa disciplina. Em termos simples, nesta técnica, a expressão lógica é escrita considerando a lógica AND como "multiplicação", e a lógica "OR" como soma. Sendo assim, por exemplo, a expressão da tabela do segmento "e" que acabamos de ver seria:

$$\overline{b3}.\overline{b2}.\overline{b1}.\overline{b0} + b3.\overline{b2}.\overline{b1}.\overline{b0} + b3.\overline{b2}.b1.\overline{b0} + b3.\overline{b2}.b1.b0$$

Esta expressão pode ser simplificada algebricamente visando obter uma expressão mais simples. No entanto, a técnica não se aproveita das saídas irrelevantes, o que limita a eficácia do método.



Outro método mais eficiente e interessante é o método de Karnaugh, que utiliza uma simplificação "gráfica" da expressão. Com base na matriz acima, onde cada célula corresponde a uma determinada combinação das variáveis de entrada, preenchemos cada uma com o valor da saída correspondente. Após isto, agrupamos os conjuntos de bits 1 de forma a que todos os bits 1 façam parte de algum grupo. Quanto maior o grupo, mais simples é a expressão; grupos com oito casas são representados por apenas uma entrada; quatro casas, um AND de duas variáveis, e duas casas, um AND de três variáveis. O resultado, no caso da expressão do segmento "e", fazendo as simplificações demonstradas na figura, é:

$$E = C . \overline{D} + \overline{B} . D$$

Veja que precisaremos apenas de duas portas NOT (para inverter C e D), duas portas NAND, e uma porta OR. Além disto, nenhuma porta tem mais de duas entradas. Temos, portanto, um circuito muito mais simples, e que cumpre a mesma função.

O programa de nossa disciplina não inclui um estudo aprofundado das técnicas de simplificação, mas para os mais interessados, o livro do IDOETA trata deste tema de forma muito mais detalhada.

15 Lógica Combinacional x Lógica Sequencial

Em todos os exemplos que vimos até agora, as saídas do circuito podiam ser obtidas a partir dos valores das entradas, obedecendo a uma tabela verdade. Em algumas situações, no entanto, é importante obter saídas determinadas não só pelos valores atuais das entradas, como também pelos valores anteriores delas.

Algumas operações básicas de um computador refletem este comportamento. Um programa de computador, por exemplo, nada mais é do que a execução **sequencial** de comandos, onde valores de variáveis são alterados muitas vezes com base nos seus valores anteriores.

Esta demanda determinou a criação de circuitos lógicos específicos, que também se utilizam das mesmas portas lógicas básicas já estudadas, mas que têm este comportamento. Na prática, o comportamento é obtido através da realimentação, ou seja, ao menos parte das saídas do circuito são conectados como entradas.

Embora não faça parte do programa da disciplina o estudo detalhado destes circuitos, é importante saber da existência deles, que permitem aplicações fundamentais em um computador, como registradores de deslocamento (utilizados, entre outras coisas, para multiplicar números binários), memórias e contadores.

Novamente, para os mais interessados, o livro do IDOETA trata deste tema de forma muito mais detalhada.

16 Estrutura de um Computador

16.1 ENIAC: o primeiro computador de uso geral

Conforme vimos, o ENIAC (*Electronic Numerical Integrator and Computer*) é um dos primeiros computadores citados na história dos computadores. Ele possuía algumas características marcantes, como por exemplo, o fato de ser baseado na base 10, e não 2, como os computadores atuais.

No entanto, a maior inovação apresentada no ENIAC era o fato dele ser considerado a primeira máquina "programável", ou seja, ele podia ser utilizado para diversas diferentes aplicações, desde que fosse preparado previamente para aquilo.

Por outro lado, como também vimos, esta programação era demasiadamente complexa, pois exigia o posicionamento de chaves e conexões elétricas via cabos, exigindo, do programador, conhecimento de sua arquitetura de *hardware*.

16.2 O IAS, a máquina de Von Neumann

Desenvolvido logo depois da criação do ENIAC, o IAS, dispositivo que levava o nome da instituição onde foi criado (*Institute for Advanced Studies*), trazia grandes avanços. Projetado por um dos construtores do ENIAC, o matemático John von Neumann, o computador possuía conceitos inovadores.

Von Neumann propôs que as instruções, que naquela época eram lidas em cartões perfurados, fossem armazenadas na memória, o que faria que sua execução fosse mais rápida. Além disto, toda a arquitetura proposta por ele é utilizada até hoje na maior parte dos computadores atuais. O IAS, portanto, foi a primeira máquina "de programa armazenado", ou seja, pela primeira vez os programas eram armazenados na memória principal do computador, assim como os dados por ele manipulados.

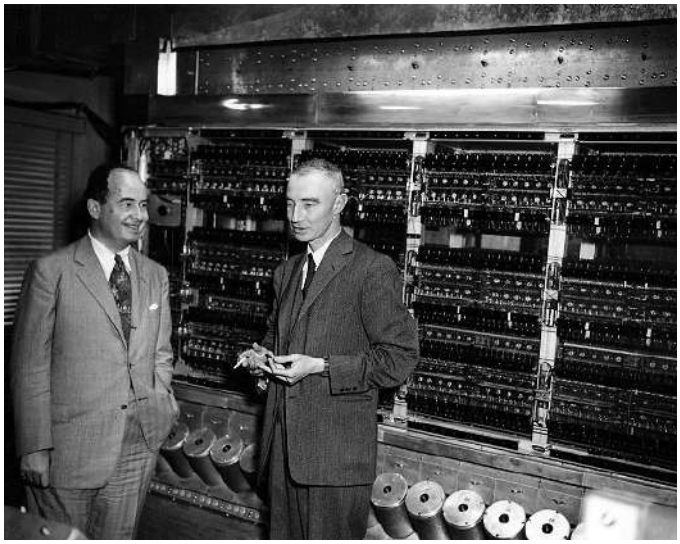
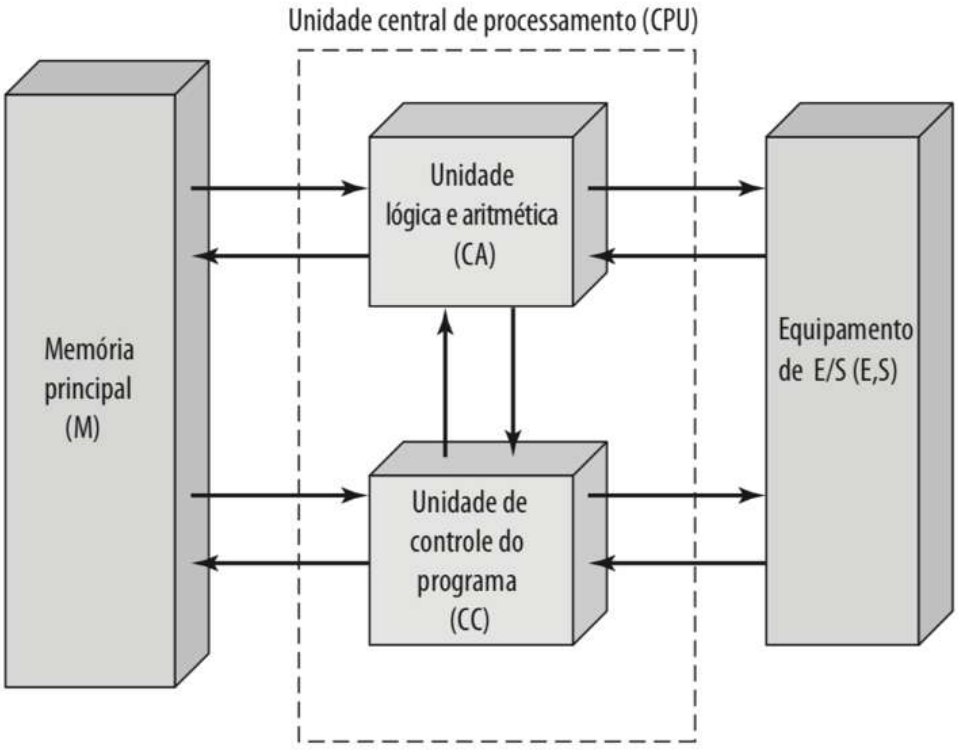
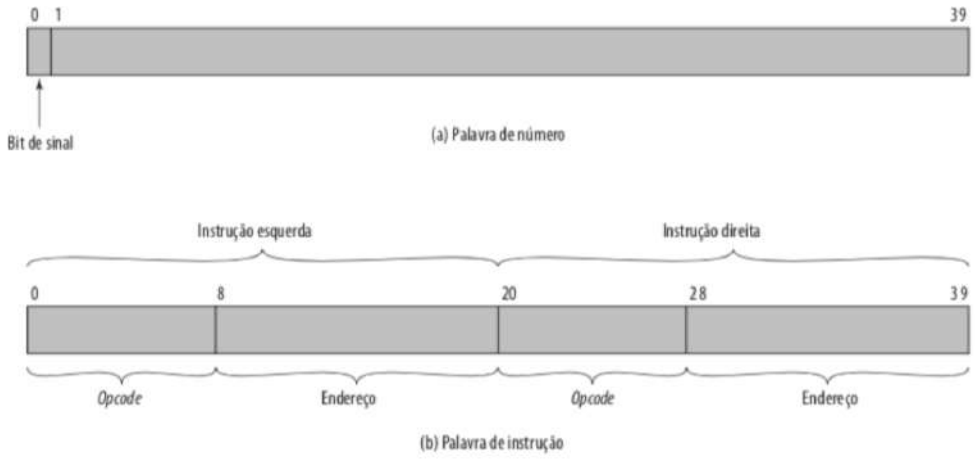


Figura 4 Von Neumann (à esquerda), com o ENIAC ao fundo, em 1952.

Sua estrutura, até hoje utilizada, envolvia os seguintes elementos:



Com 1.000 posições de memória contendo palavras de 40 bits, o IAS, até hoje conhecido como "máquina de Von Neumann" armazenava, em cada palavra de memória, um número ou caractere codificado, ou então duas instruções de 20 bits, sendo que os primeiros 8 bits continham o "OpCode" (código que identificava a instrução), e os 12 bits finais continham os operadores daquela instrução específica (tipicamente o endereço de memória afetado pela instrução).



16.3 Estrutura de Alto Nível

Um computador, tal como o conhecemos desde o surgimento da máquina de Von Neumann, pode ser dividido em 3 blocos principais: a CPU, a Memória Principal e os dispositivos de Entrada/Saída. Estes blocos são interligados por meio dos barramentos de comunicação.

16.4 Estrutura da CPU

A CPU, por sua vez, pode também ser dividida em 3 blocos principais: a ULA (Unidade Lógica e Aritmética), a Unidade de Controle e os Registradores. Desta vez, os blocos são interligados pelos barramentos internos da CPU.

A ULA é responsável pela execução de operações aritméticas como soma e multiplicação, por exemplo, além de operações lógicas como comparação, e aplicação de lógicas binárias E, OU, NÃO, XOR, teste de valores etc.

A Unidade de Controle é a "CPU da CPU". Ela é responsável por converter cada uma das instruções do programa na execução de uma sequência de operações digitais e transferência de conteúdos entre os elementos internos da CPU.

Os registradores são pequenas memórias, extremamente ágeis, que mantém informações de controle da CPU, além de armazenar resultados parciais durante cálculos da ALU. O PC (*Program Counter*), por exemplo, mantém o endereço da próxima instrução a ser executada. Já o AC (*Accumulator*) armazena resultados intermediários de cálculos, como por exemplo durante a soma de diversos valores.

16.5 Estrutura da Unidade de Controle

A Unidade de Controle por sua vez é dividida em 3 blocos: a Memória de Controle, a Lógica de Sequência e os Registradores (ambos circuitos sequenciais) e também os Decodificadores (circuitos combinacionais).

17 Operando o IAS

Para entender a operação do IAS, e por consequência de praticamente qualquer dispositivo computacional, vamos estudar a função de seus elementos, e as operações básicas realizadas por eles.

17.1 Os registradores do IAS

MBR – *Memory Buffer Register*: mantém temporariamente um valor que acaba de ser lido na memória, ou que será escrito em seguida. Funciona como um *buffer* temporário para garantir o desacoplamento entre a execução de operações comandadas pela Unidade de Controle, e o acesso à memória, que tipicamente ocorrer de forma bem mais lenta.

MAR – *Memory Address Register*: opera de forma similar ao MBR, só que para armazenar endereços de memória ou dispositivos de E/S que serão utilizados em seguida.

IR – *Instruction Register*: armazena temporariamente a instrução que está sendo executada, para que esta possa ser analisada pelos circuitos de controle.

IBR – *Instruction Buffer Register*: para acelerar a execução dos programas, este registrador antecipa a recuperação da próxima instrução a ser executada, permitindo a transferência imediata da mesma após a execução da instrução em análise.

PC – *Program Counter*: conforme já citamos anteriormente, mantém o endereço da próxima instrução a ser executada.

AC – *Accumulator*: conforme também já citamos anteriormente, armazena o resultado intermediário de cálculos com múltiplos operadores ou etapas.

MQ – *Multiplier Quotient*: tem função análoga ao AC, porém para operações de multiplicação e divisão.

17.2 Ciclo de Instrução no IAS

O IAS, assim como qualquer outro computador, executa programas instrução por instrução, como uma máquina sequencial. Para cada instrução, o computador precisa buscar, carregar e finalmente executar a instrução. Este processo, chamado de "ciclo de instrução" está dividido em dois ciclos: a busca e a execução.

No ciclo de busca, a instrução a ser executada é carregada da memória no registrador IR, assim como seus operadores e endereços secundários. Isso envolve indicar a posição de memória onde a instrução está, armazenando o mesmo no registrador MAR, por exemplo.

No ciclo de execução, a instrução é executada, e seus resultados são devidamente armazenados nos registradores e posições de memória corretos.

Usando os registradores apresentados anteriormente, apresentamos a seguir o fluxograma parcial do ciclo de instrução do IAS.

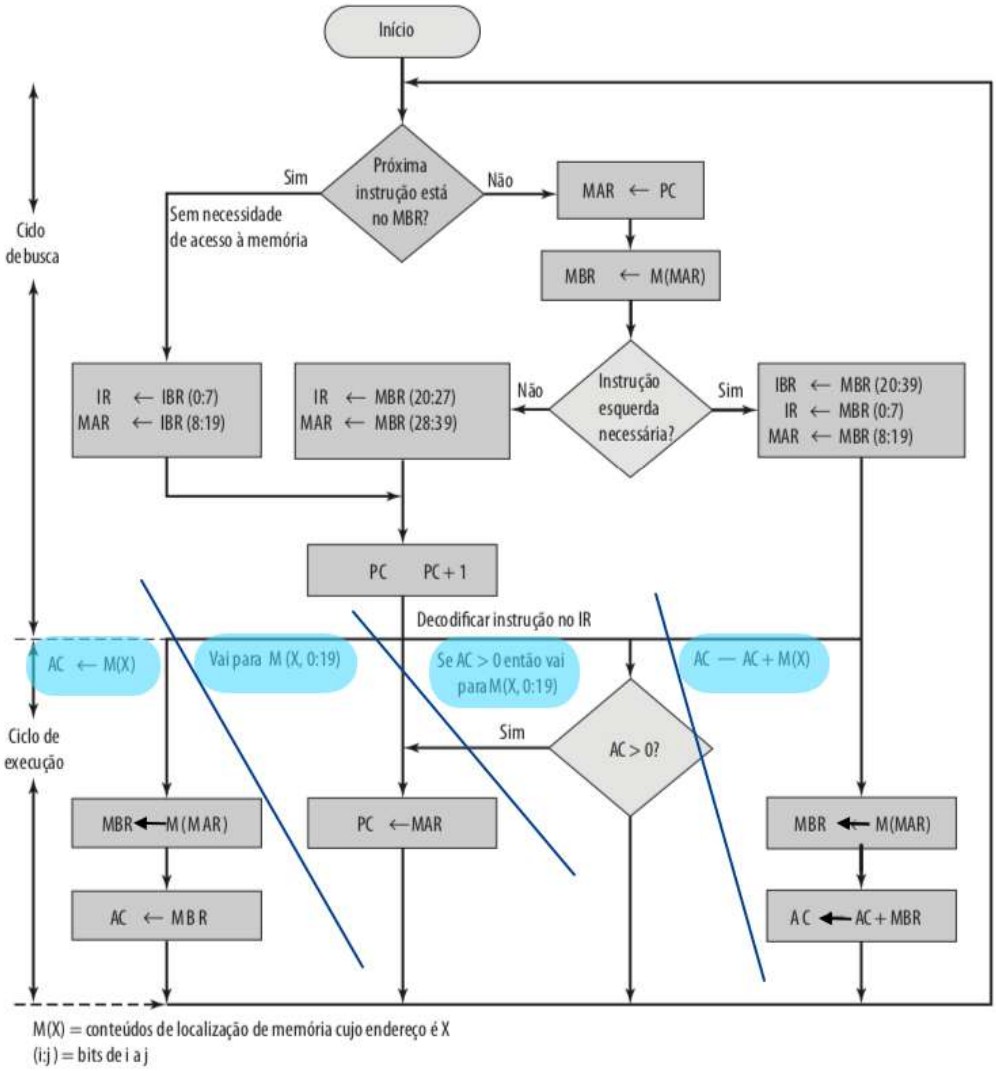


Figura 5 - Ciclos de Busca e Execução (4 instruções), modificada pelo autor [STALLINGS], Figura 2.4, P.17

Na figura anterior, para simplificar a análise, foram marcadas com fundo azul as quatro instruções exemplo que estariam sendo supostamente executadas, e separadas com traços azuis escuros as partes do fluxograma relativas à execução de cada uma delas.

Podemos ver que o fluxograma se encontra dividido em duas partes: a parte superior, que representa a "busca" e a parte inferior, que representa a "execução", neste último caso contendo quatro possíveis instruções diferentes.

17.3 O *Assembly* do IAS

Mas quais são as instruções possíveis? A tabela a seguir, disponível no livro do Stallings, apresenta as 21 instruções da linguagem Assembly do IAS. São basicamente cinco tipos de instrução:

a) Transferência de dados entre registradores e posições de memória;

b) Desvios incondicionais, que mudam a sequência padrão de execução das instruções, permitindo a repetição de trechos do programa, por exemplo;

c) Desvios condicionais, que mudam a sequência padrão de execução das instruções com base no valor do Acumulador (como vimos, ele é um dos registradores da CPU do IAS);

d) Aritméticas, que são constituídas de operações matemáticas básicas em binário;

e) Modificação de endereço, que permite a manipulação de endereços com base em resultados obtidos pela ULA.

Tipo de instrução	*Opcode*	Representação simbólica	Descrição
Transferência de dados	00001010	LOAD MQ	Transfere o conteúdo de MQ para AC
	00001001	LOAD MQ,M(X)	Transfere o conteúdo do local de memória X para MQ
	00100001	STOR M(X)	Transfere o conteúdo de AC para o local de memória X
	00000001	LOAD M(X)	Transfere M(X) para o AC
	00000010	LOAD – M(X)	Transfere – M(X) para o AC
	00000011	LOAD |M(X)|	Transfere o valor absoluto de M(X) para o AC
	00000100	LOAD – |M(X)|	Transfere -|M(X)| para o acumulador
Desvio incondicional	00001101	JUMP M(X,0:19)	Apanha a próxima instrução da metade esquerda de M(X)
	00001110	JUMP M(X,20:39)	Apanha a próxima instrução da metade direita de M(X)
Desvio condicional	00001111	JUMP+ M(X,0:19)	Se o número no AC for não negativo, apanha a próxima instrução da metade esquerda de M(X)
	00010000	JUMP + M(X,20:39)	Se o número no AC for não negativo, apanha a próxima instrução da metade direita de M(X)
Aritmética	00000101	ADD M(X)	Soma M(X) a AC; coloca o resultado em AC
	00000111	ADD |M(X)|	Soma |M(X)| a AC; coloca o resultado em AC
	00000110	SUB M(X)	Subtrai M(X) de AC; coloca o resultado em AC
	00001000	SUB |M(X)|	Subtrai |M(X)| de AC; coloca o resto em AC
	00001011	MUL M(X)	Multiplica M(X) por MQ; coloca os bits mais significativos do resultado em AC; coloca bits menos significativos em MQ
	00001100	DIV M(X)	Divide AC por M(X); coloca o quociente em MQ e o resto em AC
	00010100	LSH	Multiplica o AC por 2; ou seja, desloca à esquerda uma posição de bit
	00010101	RSH	Divide o AC por 2; ou seja, desloca uma posição à direita
Modificação de endereço	00010010	STOR M(X,8:19)	Substitui campo de endereço da esquerda em M(X) por 12 bits mais à direita de AC
	00010011	STOR M(X,28:39)	Substitui campo de endereço da direita em M(X) por 12 bits mais à direita de AC

Tabela 1 - Conjunto de instruções do IAS [STALLINGS] Tabela 2.1, p.18.

Com base nesta tabela de instruções, você seria capaz de dizer quais seriam os passos necessários para executar cada uma das instruções apresentadas na figura? A definição destas etapas internas de execução, análogas a um programa de computador, são conhecidas como microcódigo, e são "programadas" pelo fabricante da CPU. Muitas vezes a qualidade deste microcódigo é mais relevante do que o *hardware* interno do processador, motivo pelo qual este é preservado em sigilo e protegido por patentes pelo fabricante do processador.

18 Memória – características

Qualquer dispositivo de memória possui características básicas que o distingue entre as mais variadas tecnologias disponíveis para dispositivos computacionais. São elas:

18.1 Localização

Normalmente classifica-se os dispositivos de memória em internos e externos. Os dispositivos externos normalmente estão conectados ao computador através de interfaces específicas. São os discos magnéticos (HD, ou *hard disk*), unidade SSD, unidades de disco ótico etc. Já os dispositivos internos

normalmente estão conectados diretamente à CPU, tais como a memória RAM do computador, os registradores internos da CPU e o cache, por exemplo.

18.2 Capacidade

Normalmente é a característica mais observada ao selecionar um dispositivo de memória para um computador. Pode ser medida diretamente em Bytes e seus múltiplos, ou em palavras. Como a palavra de um processador determina o número de bits processado em uma instrução, processadores com palavras maiores normalmente demandam memórias maiores para um ganho de desempenho efetivo.

18.3 Unidade de Transferência

Dispositivos de memória são acessados através da leitura ou gravação de conjuntos pré-definidos de dados. O menor bloco é do tamanho da palavra do processador (número de bits, como por exemplo 64 bits para os processadores comuns hoje em dia). Outro conjunto típico é conhecido como bloco, e é utilizado em unidades de disco magnético, por exemplo.

18.4 Método de Acesso

Um dispositivo de memória pode ser acessado de forma:

Sequencial: método comum para unidades de fita, por exemplo;

Direta: na memória principal do computador, por exemplo, o item a ser lido é acessado diretamente a partir de seu endereço;

Aleatório: não existe uma sequência pré-determinada de acesso, como nos discos magnéticos, por exemplo, onde a tabela de alocação de arquivos é acessada antes do arquivo propriamente dito;

Associativo: quando se utiliza uma tabela, ou algoritmo específico para determinar a localização de uma informação dentro do dispositivo de memória.

18.5 Desempenho

O acesso à informação armazenada em um dispositivo de memória demanda um tempo de leitura que é composto por 3 componentes:

Tempo de acesso: é o tempo gasto entre a indicação do endereço da informação a ser recuperada ou gravada, e o início do acesso aos dados no dispositivo. Em dispositivos mecânicos, como unidades de disco magnético, por exemplo, este tempo pode ser bastante elevado, devido à necessidade de movimentação mecânica do cabeçote de leitura/gravação até a área onde os dados estão ou serão armazenados.

Tempo de ciclo: é o tempo gasto entre o início de acesso e o término da leitura ou escrita no dispositivo.

Taxa de transferência: é o tempo gasto para transferir os dados já recuperados, do dispositivo de memória para o computador, ou vice-versa.

18.6 Tipo Físico

Identifica a mídia onde os dados serão armazenados. Além de semicondutores como na memória RAM de um computador, temos meios magnéticos e mídia ótica, por exemplo.

18.7 Características Físicas

Algumas características físicas são importantes para identificar as propriedades de um dispositivo de memória. A volatilidade ou a impossibilidade de apagamento e reescrita são restrições importantes em algumas mídias.

18.8 Organização

Os dispositivos de memória são ofertados em diferentes formatos e organizações. Memórias RAM, por exemplo, normalmente são ofertadas em módulos de diferentes tamanhos e performances.

19 Hierarquia de Memória

Diferentes dispositivos de memória normalmente oferecem diferentes performances e capacidades de armazenamento. Normalmente estas características se comportam de forma oposta, ou seja, dispositivos com grande capacidade de armazenamento normalmente são mais lentos, e vice-versa. A figura a seguir demonstra esta hierarquia sob os pontos de vista opostos:

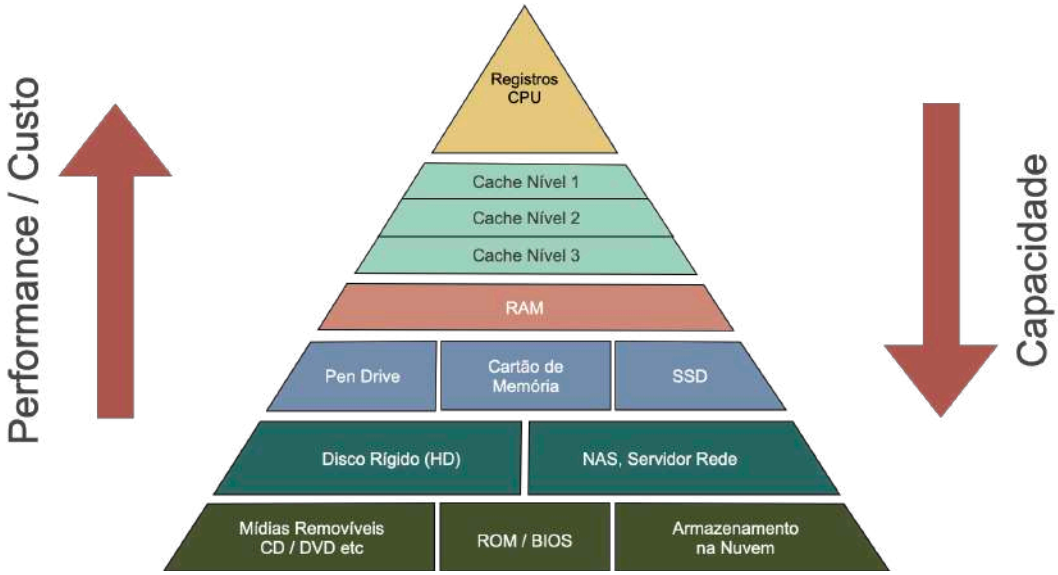


Figura 6 - Hierarquia de Memória (Autor)

No topo da figura vemos o dispositivo de memória mais rápido, e ao mesmo tempo, com a menor capacidade de armazenamento, que são os registradores da CPU. Descendo, encontramos dispositivos de memória cada vez mais lentos, porém com cada vez mais capacidade de armazenamento, como os discos rígidos, por exemplo. Por isto é necessário escolher cuidadosamente o dispositivo que armazenará cada tipo de informação, de forma a garantir uma boa relação custo/capacidade.

A tabela a seguir apresenta valores típicos de performance e capacidade de dispositivos de memória típicos.

Nível	1	2	3	4	5
Nome	registradores	cache	memória principal	disco de estado sólido	disco magnético
Tamanho típico	< 1 KB	< 16 MB	< 64 GB	< 1 TB	< 10 TB
Tecnologia de implementação	memória personalizada com várias portas CMOS	CMOS SRAM em chip ou fora do chip	CMOS SRAM	memória flash	disco magnético
Tempo de acesso (ns)	0,25 – 0,5	0,5 – 25	80 – 250	25.000 – 50.000	5.000.000
Largura de banda (MB/s)	20000 – 100000	5000 – 10000	1000 – 5000	500	20 – 150

Tabela 7 – Comparativo de Performance dos dispositivos de memória. [SILBERCHATZ] Figura 1.11, Página 39

20 A memória cache

Um dos tipos específicos de memória instalados em dispositivos computacionais é a chamada Memória Cache. Seu objetivo é servir de intermediária entre o processador e a memória principal do computador. Como se trata de uma memória de alta performance, a antecipação da transferência de conteúdos de memória para ela acaba aumentando a performance do computador.

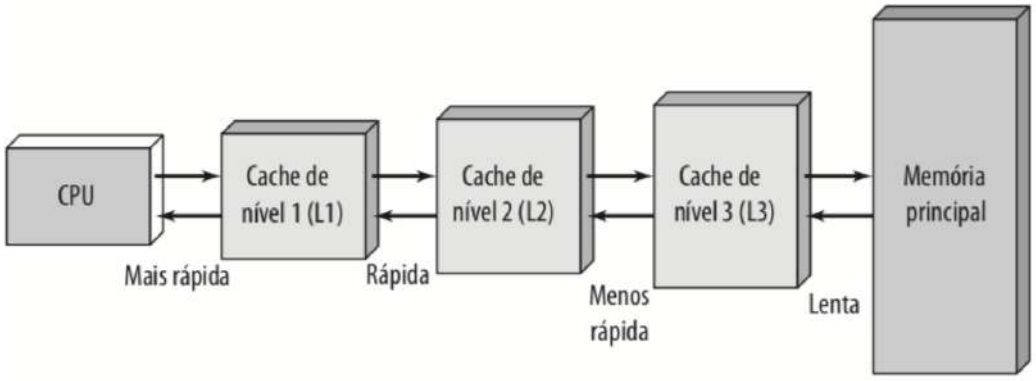


Figura 8 - Organização da memória cache em níveis. [STALLINGS], Fig.4.3 P.96

A memória cache se classifica em "níveis", onde os níveis iniciais, identificados pelos menores números, ficam mais próximos da CPU, e tipicamente são mais rápidos e menores. Os níveis mais altos normalmente estão instalados um pouco mais distantes da CPU, e, portanto, são maiores, e mais lentos - embora ainda bem mais rápidos do que o acesso direto à memória principal.

Ao longo da história dos dispositivos, a memória cache tem evoluído tanto no número de níveis quanto na sua capacidade e performance. Um computador extremamente comum na década de 1970, como o IBM 360/85, possuía apenas um nível de cache (L1), com capacidade variável entre 16 e 32KB. Embora se tratasse de um computador "de grande porte", a evolução ao longo de 40 anos nos trouxe, desta vez em um microprocessador, o Intel i7 (até hoje muito comum), para 3 níveis de cache, todos incluídos na pastilha semicondutora. A capacidade de L1 subiu para

até 24 blocos com 128KB (100x), além de um espaço também com 24 blocos de 1,5MB em L2, e até 24MB em L3. O i7 também suporta até 192MB de L4.

Ou seja, o aumento de performance obtido com este dispositivo tornou o seu uso cada vez mais comum nos computadores modernos, que têm cada vez mais níveis de memória cache com capacidades cada vez maiores. Mesmo assim, como sua capacidade é muito menor que a capacidade da memória principal, são necessários algoritmos sofisticados para escolher qual conteúdo deve ser transferido da memória para o cache, de forma a acelerar o desempenho do computador. O objetivo é evitar ao máximo o acesso direto do processador à memória principal, deixando para o *hardware* controlador de memória as operações que envolvem a "lenta" memória convencional.

A memória cache é utilizada tanto para a leitura quanto para a escrita na memória principal. No primeiro caso, o seu uso não acarreta riscos, já que todo o conteúdo acessado também está disponível na memória principal. No entanto, nas operações de escrita, é importante garantir a atualização da memória principal assim que possível, uma vez que a falta de atualização pode até provocar a existência de conteúdos diferentes entre dois processos ou *threads*, para a mesma posição de memória. Esta inconsistência pode provocar falhas graves na execução de programas, e, portanto, precisa ser devidamente eliminada.

21 Barramentos

Os barramentos são estruturas de condutores, normalmente gerenciadas por dispositivos digitais específicos, para a interligação dos componentes de um computador. São três os barramentos utilizados em um dispositivo computacional baseado no conceito da máquina de Von Neumann:

Barramento de Endereços: compartilha a identificação do endereço da posição de memória ou dispositivo de entrada/saída a ser acessado em determinado momento. Escrito pela CPU, o barramento de endereços precisa ser lido simultaneamente por todas as posições de memória e todos os dispositivos de entrada e saída.

Barramento de Dados: transfere informações a serem lidas ou escritas em posições de memória ou dispositivos de entrada e saída pela CPU. Neste barramento, o fluxo de dados é bidirecional, pois cada um dos 3 componentes de um computador pode ser lido ou escrito.

Barramento de Controle: contém diversos sinais de controle operacional sobre os diversos dispositivos de um computador. Como exemplo, podemos citar o R/W. Este sinal determina para a memória ou dispositivo de entrada e saída se, naquele momento, ele será escrito ou lido. A depender do sinal específico, o fluxo de dados pode ocorrer de forma unidirecional ou bidirecional. O barramento, então, é bidirecional, já que precisamos incluir nele diversos tipos de sinal diferentes.

22 Máquinas Paralelas

Além da arquitetura padrão proposta pelo Von Neumann, foram desenvolvidos outros modelos de arquitetura para dispositivos computacionais avançados.

Um dos problemas que desejava-se contornar era a superioridade de performance do processador (CPU) em relação aos processos de recuperação ou armazenamento de informações nos outros dispositivos, como a memória principal, através dos barramentos.

Para contornar este problema, além do uso de dispositivos como a memória cache, foram discutidas e estudadas outras arquiteturas, classificadas por Michael J. Flynn, professor da universidade de Stanford.

		Fluxo de Dados	
		Único	Múltiplo
Fluxo de Instruções	Único	SISD Single Instruction Single Data Instrução Simples de Dados Simples	SIMD Single Instruction Multiple Data Instrução Simples de Múltiplos Dados
	Múltiplo	MISD Multiple Instruction Single Data Instrução Múltiplas de Dados Simples	MIMD Multiple Instruction Multiple Data Instrução Múltiplas de Dados Múltiplos

Figura 9 – Taxonomia de Flynn. Fonte: Wikipedia

A taxonomia de Flynn propõe a classificação das arquiteturas de dispositivos computacionais em quatro tipos básicos.

22.1 SISD

A arquitetura SISD inclui a máquina de Von Neumann, e também algumas tecnologias derivadas da mesma, como os processadores de Pipeline e Superescalares.

Processadores de Pipeline levam em consideração a execução das instruções em passos sequenciais, e assim distribuem diversas instruções pelas diversas partes do processador, tentando manter todas as partes ocupadas ao mesmo tempo, para ganhar performance.

Processadores superescalares executam mais de uma instrução simultaneamente, através da distribuição de instruções para diversas unidades do processador.

22.2 SIMD

A arquitetura SIMD se baseia em aplicações onde há paralelismo de dados, mas não há paralelismo de instruções. Algumas se beneficiam especificamente disto, como as placas gráficas, que processam volumes significativos de dados vetoriais.

22.3 MISD

Esta arquitetura é raramente implementada, embora alguns classifiquem sistemas *pipeline* dentro desta categoria. Encontramos raros exemplos em sistema de controle muito complexos onde os dados precisam ser analisados simultaneamente e em alta performance (aeroespaciais, por exemplo).

22.4 MIMD

Esta arquitetura é a base dos chamados sistemas distribuídos, onde diversos dispositivos computacionais, tipicamente interligados através de uma rede de comunicação de dados, trocam informações de forma a dividir a execução de um processo para ganho de desempenho.

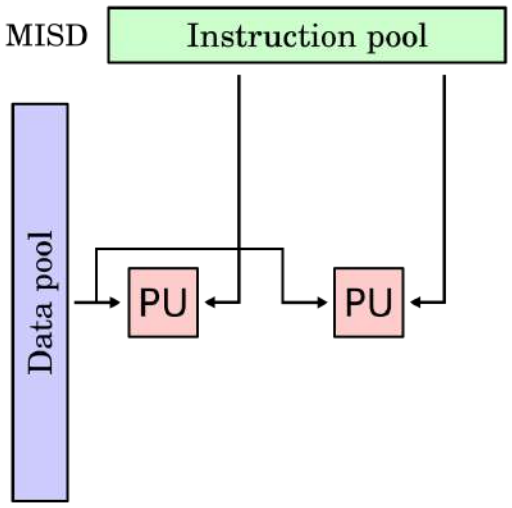


Figura 10 - Modelo Operacional da Arquitetura MISD [Wikipedia MISD]

22.5 Arquitetura *Pipeline*

Um exemplo de arquitetura paralela adotada em algumas aplicações atuais é processamento *Pipeline*.

Antes de começarmos, vamos usar uma analogia para explicar o conceito básico de um sistema *pipeline*. Diversas atividades humanas, e também processos computacionais, envolvem etapas bem claras e definidas, que precisam ser executadas em sequência para atingirmos um resultado esperado.

Considerando um serviço comunitário de lavanderia em um condomínio, poderíamos identificar 3 etapas claras, independentes e sequenciais: a lavagem, a secagem e a dobra das peças depois de lavadas e secas. Considerando, a título de exemplo, um período de 30 minutos para completar uma lavagem, seguido de um período de 40 minutos para a secagem, e mais 20 minutos para a dobra das peças, seria necessário uma hora e meia para uma pessoa concluir o trabalho.

Em um serviço comunitário, todos os moradores precisam compartilhar os recursos de lavagem, secagem, e o espaço para a dobra, de forma que todos possam realizar a atividade de forma organizada. Temos, a princípio, duas estratégias possíveis:

- Na primeira estratégia, cada condômino tem todos os recursos à sua disposição, e só precisa liberar os mesmos após concluir o seu trabalho. Exemplificando, supondo a presença de quatro pessoas, teríamos um tempo total de seis horas para conclusão da atividade de todos.

- Em uma segunda estratégia – adotando um esquema *pipeline* - o primeiro a utilizar os recursos liberaria a máquina de lavar para o segundo vizinho enquanto cuidava da secagem de suas peças. Da mesma forma, ao concluir a secagem, liberaria a máquina secadora para o vizinho que estava lavando suas peças, ao mesmo tempo que este liberaria a máquina de lavar para o terceiro, e assim sucessivamente. Supondo novamente a presença de quatro pessoas, teríamos um tempo total de três horas e meia para conclusão da atividade de todos!

A estratégia *pipeline*, então, oferece clara vantagem "comunitária", embora não reduza o tempo necessário para cada condômino.

Em um sistema computacional similar, poderíamos por exemplo separar as etapas de busca e execução no fluxograma de um processador, permitindo acelerar o tempo total de execução de um código. Esta

estratégia é utilizada, por exemplo, nos microcontroladores. Para isto, eles não seguem o conceito da máquina de Von Neumann, possuindo múltiplas áreas independentes de memória, e barramentos. Como programa e dados ficam em áreas separadas, é possível se dedicar à busca de uma instrução ao mesmo tempo em que processamos uma instrução anterior, manipulando os dados em outra área de memória, acelerando o tempo de execução total de boa parte dos códigos de aplicações.

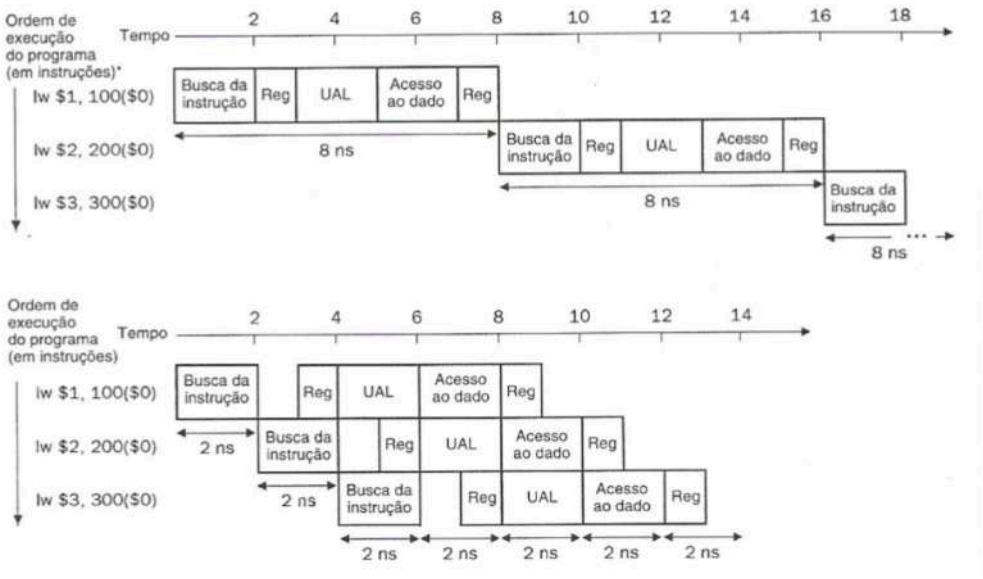


Figura 11 - Aumento de eficiência aplicando pipeline aos processos de busca e execução de uma instrução.

22.6 Arquitetura superescalar

Tipicamente classificada como uma evolução do *pipeline*, sistemas baseados nesta arquitetura são capazes de executar múltiplas instruções simultaneamente, sem que estas estejam obrigatoriamente em estágios de *pipeline* diferentes.

Para isto, são necessárias mudanças ainda mais profundas na arquitetura de hardware, incluindo a criação de duas novas unidades funcionais, a Unidade de Busca de Instruções, e a Unidade de Decodificação.

A Unidade de Busca de instruções realiza a busca simultânea de múltiplas instruções, retirando este trabalho da Unidade de Controle da CPU. Para isto ela inclusive tenta descobrir o fluxo resultante dos próximos desvios, com base em um algoritmo preditivo, de forma a garantir a busca das instruções corretas.

A Unidade de Decodificação faz a leitura simultânea dos diversos registradores de instrução, garantindo a execução simultânea das diversas instruções lidas pela Unidade de Busca.

Outra mudança importante é a presença de múltiplas ULAs e unidades aritméticas de ponto flutuante, também para garantir a execução simultânea.

23 Computação Física

A interação do usuário com dispositivos computacionais normalmente acontece pelos dispositivos de entrada e saída convencionais, como teclado, telas sensíveis ao toque, mouse e dispositivo apontadores, microfones e alto-falantes. No entanto, o poder dos dispositivos computacionais pode ser aplicado diretamente em soluções “físicas" no mundo real, com medição e sensibilidade a eventos físicos, e com a atuação sobre eles, sem intervenção humana.

Dispositivos computacionais que fazem este trabalho podem, muitas vezes, ser mais simples e de baixo custo. É o caso dos dispositivos baseados em microcontroladores, como as placas de protótipo Arduíno, por exemplo.

Este novo segmento, que mistura computação e eletrônica, levou a automação para o dia a dia das pessoas e organizações, e é conhecida como "Computação Física".

23.1 Microcontroladores X Microprocessadores

Já estamos acostumados com o conceito de microprocessador, já que ele é a CPU presente nos dispositivos computacionais de nosso dia a dia. Alguns usuários até identificam fabricantes e modelos, como por exemplo o Intel i7 e o Apple M3 em computadores, ou o A15 Bionic e o Snapdragon 8 em celulares. Seus recursos e performances muitas vezes determinam a escolha de um produto específico, ou das aplicações compatíveis.

No entanto, ao contrário do que pode parecer em uma análise mais superficial, nem todos os dispositivos com capacidade embutida de "processamento" possui um microprocessador. Existe uma outra classe de CPUs, desenvolvidas para aplicações de computação física. Estamos falando dos **microcontroladores**.

Microcontroladores são desenvolvidos para ficarem "embutidos" em equipamentos, ferramentas e máquinas, muitas vezes literalmente escondidos de seus usuários. Um motorista, por exemplo, pode usar seu automóvel durante toda a vida útil sem sequer tomar conhecimento da existência dos múltiplos microcontroladores presentes no mesmo. Este é conceito de "sistema embarcado", que, quando se utiliza de recursos de comunicação via Internet, assume o nome de IoT (*Internet of Things* – Internet das Coisas).

O desenvolvimento dos microprocessadores tipicamente tem como objetivo a obtenção de números cada vez melhores de performance, capacidade de memória, acesso a periféricos e recursos gráficos. Já nos microcontroladores, os objetivos são redução das dimensões, consumo energético e custo. Com isto, eles incorporam funcionalidades ausentes nos microprocessadores, como memória embutida, ADC (conversor analógico/digital), controladores embutidos de rede etc.

Para facilitar a integração com dispositivos físicos, eles também oferecem facilidades de conexão externa através de interfaces digitais e analógicas. Estas permitem integrar sensores dos mais diversos tipos, além de acionar indiretamente motores e outros equipamentos.

Todos estes recursos são controlados através de softwares relativamente simples, tipicamente baseados em uma linguagem de programação de alto nível conhecida, como o C++ e o Python, por exemplo. Os programas podem ser desenvolvidos em ambientes de desenvolvimento normalmente gratuitos. Por outro lado, como normalmente os programas executados nestes processadores, também chamados de *sketchs*, são pequenos e exigem poucos recursos computacionais, os microcontroladores também oferecem recursos computacionais modestos, como pouca memória e performance de processamento limitada, e normalmente sequer executam um sistema operacional. Isto, no entanto, não impede a criação de projetos dos mais diferentes níveis de complexidade, totalmente baseados em microcontroladores.

23.2 Aplicações típicas da Computação Física

Graças à flexibilidade de *hardware* e *software*, os dispositivos de computação física podem ser utilizados nas mais diversas aplicações, limitadas apenas pela imaginação do projetista.

Entre outras aplicações, são comuns as soluções de segurança patrimonial para residências, onde sensores de presença, de abertura de portas e janelas podem ser utilizados como entrada para sistemas que atuam na simulação de presença e no acionamento de alarmes, por exemplo.

Ainda nas residências, outra aplicação comum é a automação residencial, onde estes dispositivos podem controlar, entre outras coisas, a iluminação, sonorização e climatização, através de interação por voz e roteiros pré-programados.

Soluções residenciais, no entanto, não são as únicas que podem ser embarcadas nestes dispositivos. Eles podem ser adotados em qualquer solução onde o controle de dispositivos elétricos e mecânicos possa ser realizado de forma automática, gerando redução de consumo energético, maior segurança para os operadores humanos, proteção dos equipamentos contra danos etc. Isto vale para a automação industrial, indústria automotiva etc.

Figura 12 - Robô de limpeza.

Outro segmento que vem adotando soluções baseadas em microcontroladores é a robótica. Robôs de limpeza autônoma, assistentes pessoais para idosos e dispositivos ainda mais complexos vêm sendo oferecidos no mercado. As pesquisas avançam, o que torna muito interessante o conhecimento dos recursos da computação física, especialmente por estudantes de engenharia de *software*, analistas de sistemas e engenheiros eletricistas.

24 *Hardware* para Computação Física

Soluções de computação física envolve sensores, atuadores, dispositivos de rede, fontes de alimentação, e, é claro, microcontroladores. Diferente de projetos de *software* típicos, é interessante que o projetista tenha algum conhecimento destes dispositivos, e até mesmo conceitos de eletrônica básica.

24.1 Microcontroladores

Há algumas décadas temos fabricantes dedicados ao desenvolvimento de diversas famílias de microcontroladores, com variedade de modelos, dimensões, consumo e recursos. Um dos passos mais importantes em um projeto é a escolha do microcontrolador adequado. Eis alguns exemplos:

Intel: fabricante extremamente conhecido de microprocessadores, a Intel também foi responsável pelo lançamento, nos anos 80, de um dos microcontroladores mais conhecidos do mundo, o Intel 8051. O produto fazia parte de uma família com outros microcontroladores menos conhecidos, que foi descontinuada pela Intel. Outros fabricantes até hoje produzem produtos "similares".

Microchip Tecnology Inc.: é a empresa responsável pela linha PIC de microcontroladores, largamente utilizada em diversos sistemas de automação industrial, automotiva, dispositivos médicos e eletrônicos de consumo. Entre os mais populares, está o PIC16F877, com versões de 8 e 16 bits e conjunto de instruções bem pequeno (apenas 35 instruções). A empresa fornece também modelos bem poderosos, como o PIC18F452, com 34 portas de E/S, suporte a comunicação I2C, USART e SPI, modo de operação *stand-by* com apenas 0,2 μA, entre outros recursos. Há também produtos mais simples, como o PIC12F675, com um encapsulamento bem menor (1cm de comprimento), apenas 64B de memória RAM e 5 portas de E/S.

Além disto, em 2016 a Microchip incorporou os microcontroladores da antiga Atmel, que incluem, entre outros, o ATMega328P, utilizado no famoso Arduíno Uno.

Expressif Systems: é a empresa responsável pelas linhas de placas de protótipo ESP8266 e ESP32. Ambas são extremamente populares, especialmente por incorporarem interfaces de rede Wi-Fi e Bluetooth, além de recursos bem avançados no caso da linha ESP32 (2 núcleos, 34 portas de E/S, ADC de 12 bits, 520KB de memória RAM). O baixo custo também é um dos motivos da sua popularidade.

Figura 13 - Placa de Protótipo ESP32

A linha ESP8266 utiliza o microcontrolador L106, baseado no 106Micro desenvolvido pela Tensilica, atualmente Cadence Systems. Já o ESP32 utiliza, a depender do modelo, os microcontroladores *dual-Core* Xtensa LX6, LX7, ou um *single-core* RISC-V. O ESP32 é fabricado pela TSMC com tecnologia de 40nm.

24.2 Placas de Protótipo

Apesar dos recursos disponíveis em um microcontrolador, que são muito mais adequados a um projeto de computação física, ainda é necessário dispor de componentes externos para fazer o projeto funcionar. Para simplificar o desenvolvimento de projetos, foram desenvolvidas placas de circuito impresso contendo não só o microcontrolador, mas também estes circuitos complementares básicos. Estas placas são chamadas de "placas de protótipo".

Placas de protótipo, na prática, incluem o microcontrolador, e os componentes eletrônicos acessórios para sua operação, como cristais de quartzo, reguladores de tensão, e conectores externos para acesso a interfaces digitais e analógicas, por exemplo. Algumas placas também incluem outros componentes, como controladores especializados em comunicação USB, relógios de tempo real, interfaces de rede sem fio etc. Por outro lado, uma placa de protótipo pode não ser adequada para projetos definitivos, daí o seu nome. Isso porque conexões provisórias para testes, feitas em barras de conectores, são instáveis, e a ocupação de espaço é muito maior. Além disto, a própria fixação da placa, e das suas interfaces a um equipamento definitivo pode ser problemática.

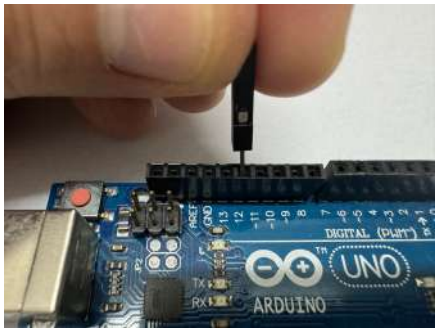


Figura 14 - Conexão de *jumper* a um Arduíno Uno

Um exemplo muito popular é a plataforma Arduíno. Ela oferece placas de protótipo em diversas versões e capacidades, em função da complexidade e recursos necessários no projeto desejado. As placas mais famosas, como o Arduíno Uno, e o Arduíno Nano, por exemplo, são utilizadas por milhares de pessoas ao redor de todo o mundo. Falaremos melhor da linha Arduíno mais tarde.

Outro exemplo bastante utilizado atualmente são as placas de protótipo da linha ESP32. Disponíveis em diversos modelos, as placas ESP32 oferecem performance e capacidade bem superiores aos produtos da linha Arduíno mais conhecidos.

Como são baseadas em microcontroladores, as placas de protótipo oferecem recursos computacionais limitados, o que pode impedir o uso em aplicações mais sofisticadas, que executem algoritmos mais complexos, ou que precisem lidar com grandes quantidades de informação. Para isto existem os chamados Computadores de Placa Única.

24.3 SBC (Single Board Computers)

Similares às placas de protótipo, os SBCs permitem conexões com sensores e atuadores externos, porém oferecem processadores mais poderosos, maior capacidade de armazenamento, e compatibilidade com dispositivos periféricos comuns em computadores convencionais, como teclados, unidades de disco e monitores de vídeo.

Figura 15 - Raspberry Pi 4B, 8GB

Um exemplo muito popular é o Raspberry Pi. Disponível em diversas versões e capacidades, o Raspberry Pi é capaz de executar aplicações complexas, que são executadas por um sistema operacional instalado no dispositivo. O Raspibian, um versão do Debian Linux desenvolvida especificamente para o Raspberry Pi, é o mais popular, embora existam outras opções, inclusive uma versão específica do Windows para aplicações IoT.

Um aspecto negativo é que, devido aos seus recursos mais avançados, os SBCs normalmente são dispositivos de custo mais elevado, além de consumirem mais energia e serem maiores do que as placas de protótipo.

25 O Arduíno Uno

A placa de protótipo Arduíno Uno é, sem dúvidas, o dispositivo de computação física mais conhecido e adotado no mundo.

O projeto Arduíno foi desenvolvido no programa do Instituto de Design Interativo Ivrea (IDII), na cidade de Ivrea, Itália. Fundado pela Telecom Itália e Olivetti em junho de 2000, o programa durou 5 anos, e rendeu, entre outros frutos, o projeto Arduíno, do qual a placa Arduíno Uno é o produto mais conhecido.

O projeto passou por evoluções, mas o Arduíno Uno sempre foi uma tecnologia aberta. A versão 3, a mais famosa, é hoje fabricada por dezenas de empresas. Isso acabou por reduzir o seu custo, com versões genéricas da placa disponíveis no mercado por menos de US$ 10 a unidade.

Quanto ao microcontrolador, o Arduíno Uno R3 (foto acima) é baseado no microcontrolador ATMega328P, possui 1KB de EPROM, 2 KB de SRAM, e 32 KB de memória Flash. Versões mais recentes, como a 4, utilizam outro microcontrolador, o Renesas RA4M1 (Arm® Cortex®-M4).

Assim como em outras placas de protótipo, para utilizar o Arduíno Uno, basta que o usuário possua um computador pessoal, já que a mesma pode ser alimentada por uma interface USB. Será também através desta porta que o usuário poderá transferir os programas específicos (também chamados de *sketches*) desenvolvidos e compilados no computador. Para aplicações independentes do computador, ou portáteis/móveis, pode ser utilizada uma fonte de alimentação externa de baixa potência (300mA) com tensão entre 5 e 17V. Esta flexibilidade na faixa de tensões de alimentação permite inclusive o uso de pilhas convencionais para projetos móveis (robôs e drones, por exemplo).

Ainda para garantir a simplicidade em pequenos projetos, o próprio Arduíno pode alimentar pequenos sensores e dispositivos externos através de saídas de 5 e 3,3V embutidas, desde que se tome o cuidado de obedecer à capacidade máxima de corrente da placa (40mA).

Como interfaces para conexão com o mundo externo, a placa oferece até 14 pinos para conexão de sensores ou saídas digitais configuráveis pelo usuário, além de 6 pinos para sensores de entrada analógicos.

25.1 Portas Digitais

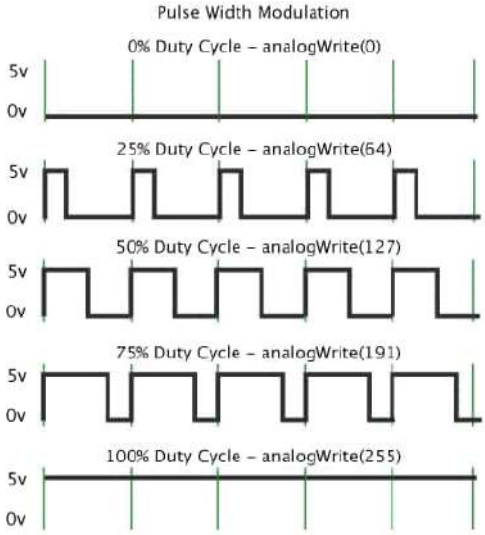
As 14 portas digitais (D0 a D13) do Arduíno Uno podem funcionar em 3 modos diferentes. Os dois modos básicos (entrada ou saída digital) podem ser selecionados para qualquer uma das 14 portas.

Quanto configuradas como entrada, é possível ler informações digitais sob o formato de tensão elétrica (0 = zero volts e 1 = cinco volts) em cada uma destas portas. Estas tensões, injetadas por sensores externos, como botões, detectores de abertura de portas etc, permitem a obtenção direta de informações do mundo físico externo.

Quando configuradas como saídas, é possível fornecer tensões elétricas representando informações digitais em cada uma destas portas. Estas tensões podem, direta ou indiretamente, acionar dispositivos externos como lâmpadas, motores, emissores sonoros etc, fornecendo resultados e informações ao mundo físico externo.

No caso específico das portas D3, D5, D6, D9, D10 e D11, temos um terceiro modo que permite obter uma simulação de "saídas analógicas". Isto é obtido através de sinais PWM (*Pulse Width Modulation*), que podem ser gerados pelo Arduíno Uno apenas nestas portas específicas, quando devidamente configurado.

Uma porta configurada desta forma gera um sinal alternado quadrado (que varia entre os estados 0 e 1, como vemos na figura). Outro nome muito comum para este tipo de sinal, comum em dispositivos computacional, é "onda quadrada".



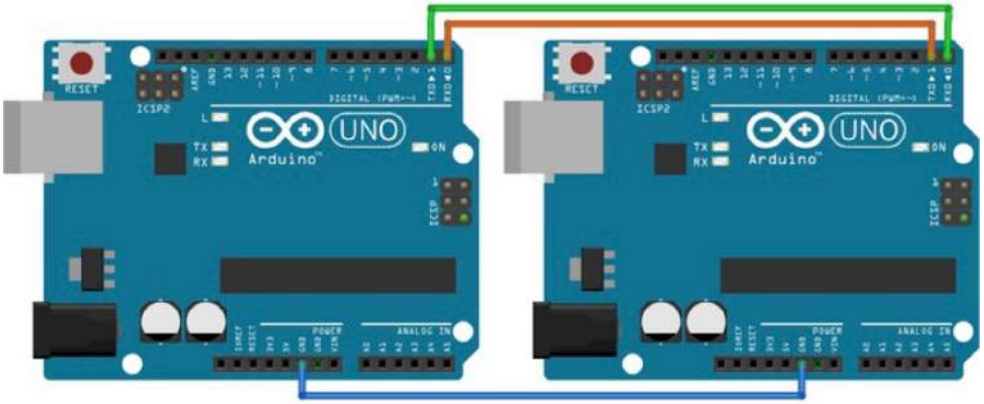
Além de determinar uma saída alternada, e não contínua, o uso do PWM permite definir o "ciclo de trabalho" (*duty cycle*) para cada porta, ou seja, o percentual de tempo em que o sinal de saída estará em nível alto (1). Com isso, podemos regular de forma precisa a potência entregue a uma carga conectada direta ou indiretamente à porta, considerando que a energia é entregue à carga apenas quando a saída está em "1". Isto é que determina a analogia com uma "saída analógica", já que o usuário pode escolher ciclos de trabalho entre 0 e 100%, com até 256 níveis diferentes (através de um parâmetro de 8 bits informado no *sketch*).

Para ficar mais claro, a figura mostra o formato da onda quadrada para diferentes valores de ciclo de trabalho. Uma lâmpada conectada a uma saída PWM, por exemplo, teria seu brilho afetado diretamente pelo valor do ciclo de trabalho estipulado na saída.

Além destes 3 modos, algumas das portas podem assumir funções especiais, que a rigor não podem ser categorizadas como "portas digitais".

25.2 Portas D0 e D1:

Estas portas normalmente ficam reservadas para a comunicação entre o Arduíno e o computador (portas de comunicação UART - *Universal Asynchronous Receiver/Transmitter*). A porta 0 é o pino de recepção (Rx) enquanto a porta 1 é o pino de transmissão (Tx). Utilizando estas portas é possível estabelecer uma comunicação entre o Arduíno Uno e outros dispositivos, inclusive com um outro Arduíno Uno, como vemos na figura.



Sendo assim, se por exemplo tivermos o Arduíno Uno conectado ao computador pela porta USB, não é recomendável utilizar a mesma para comunicação com o computador ao mesmo tempo em que as portas 0 e 1 estiverem sendo utilizadas para outra comunicação (por exemplo, com outro dispositivo).

25.3 Portas D2 e D3:

Estas são portas que podem ser habilitadas para receber sinais de interrupção. Cada uma delas funciona como um canal independente (INT0 na porta 2 e INT1 na porta 3).

Os canais de interrupção são recursos amplamente utilizados nos dispositivos computacionais. Basicamente, são responsáveis por receber sinais externos que funcionam como "gatilho" para acionamento de partes específicas de um programa (neste caso, de um "*sketch*"). Você pode, por exemplo, fazer o Arduíno Uno interromper a execução do código em um determinado ponto, e passar a executar uma parte específica do *sketch*, caso um botão conectado a uma destas duas portas for acionado. Após a execução do código da "interrupção", o programa retoma a execução no ponto onde havia parado.

Nem toda aplicação de Computação Física exige o uso de rotinas de interrupção. Nestes casos, as portas podem ser utilizadas normalmente como entradas/saídas digitais, como qualquer outra.

25.4 Portas D10 a D13:

Este conjunto de portas pode ser configurado para interligação com dispositivos que utilizem o padrão de redes SPI (*Serial Peripheral Interface*). Utilizando estas portas, o Arduíno pode se tornar um dispositivo de rede conectado a diversos periféricos. O protocolo SPI oferece performance elevada para aplicações de computação embarcada.

No Arduíno Uno R3, a conexão com uma rede SPI deve ser feita utilizando as portas 10 a 13, configuradas respectivamente como SS, MOS, MISO e SCLK*.

Nem toda aplicação de Computação Física exige a utilização de recursos de rede SPI. Nestes casos, as portas podem ser utilizadas normalmente como entradas/saídas digitais, como qualquer outra.

Outra aplicação para as portas SPI é a programação direta no circuito do dispositivo (ISP – *In-System Programming*). Neste caso, não utilizamos a porta USB ou interface UART da placa de protótipo, e podemos até dispensar o programa de *bootloader*. Esta, no entanto, é uma aplicação avançada para o escopo desta disciplina.

* O padrão SPI, cujo estudo está além do escopo desta disciplina, funciona através da conexão de dispositivos diversos (é possível conectar dezenas deles) através de quatro fios identificados como SS - Slave Select, MOSI - Master Out Slave In, MISO - Master In Slave Out e SCLK - Serial Clock. . Criado pela Motorola, acabou se tornando um padrão de fato, e é muito comum utilizá-lo na interligação com dispositivos externos de memória e mostradores digitais. Você pode encontrar maiores detalhes, por exemplo, no Wikipedia (https://pt.wikipedia.org/wiki/Serial_Peripheral_Interface). No entanto, utilizá-lo na prática é relativamente simples, devido à existência de farta documentação e diversas bibliotecas na internet.

25.5 Porta 13:

Esta porta, além de funcionar como pino SCLK para redes SPI, também está interligada a um LED instalado na placa. Sendo assim, quando ela estiver em nível 1 (*high*), o LED acenderá. Você pode utilizar o LED interno da placa para sinalizar algum status, por exemplo, sem a necessidade de instalar um LED externo.

25.6 Portas Analógicas

O Arduíno Uno R3 oferece 6 entradas analógicas A0 a A5 permitindo a leitura de até 6 valores analógicos.

Estas entradas são conectadas internamente a um ADC (Conversor Analógico-Digital) de 10 bits. Sendo assim, na configuração padrão, valores de tensão entre 0 e 5V injetados em qualquer uma das portas terá seu valor convertido para um número de 10 bits, ou seja, entre 0 e 1023.

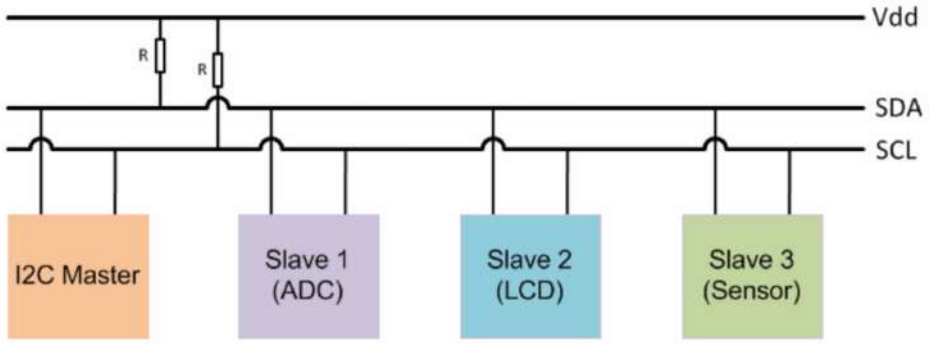
Embora uma grandeza analógica real possa assumir infinitos valores, identificar 1024 diferentes valores para uma entrada é uma boa aproximação para uma "entrada analógica". Sendo assim, embora não permitam aplicações de alta precisão, estes 10 bits são normalmente suficientes para milhares de aplicações comuns. Se, em uma determinada aplicação, for necessário ler informações analógicas com ainda mais precisão, é possível conectar interfaces externas equipadas com conversores ADC de 12, 16 e até 24 bits de precisão.

Além disto, assim como nas portas digitais, duas das seis portas analógicas podem assumir uma função especial que a rigor não pode ser categorizada como "porta analógica".

25.7 Portas A4 e A5

Estas duas portas podem ser configuradas para interligação com dispositivos que utilizem o padrão de redes i2C. Assim como o SPI, o padrão i2C também permite interligar periféricos ao Arduíno Uno R3.

A figura mostra um exemplo de rede i2C, com um dispositivo "mestre" (que pode ser o próprio Arduíno Uno, e 3 "escravos" com funções específicas (ADC externo, display LCD e sensor). Todos encontram-se interligados aos mesmos fios SDA e SCL.

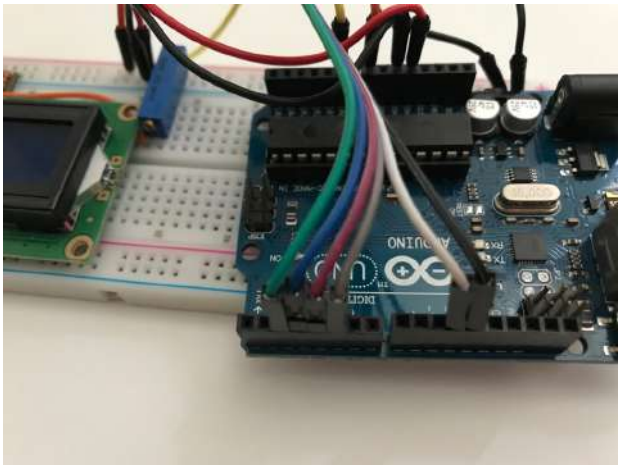


No Arduíno Uno R3, as portas A4 e A5 podem ser configuradas respectivamente como portas SDA e SCL*. O padrão permite a conexão de dezenas de dispositivos através apenas destes dois fios, ampliando muito a capacidade de conexão de sensores e outros dispositivos à placa.

* O padrão i2C, cujo estudo está além do escopo desta disciplina, funciona através da conexão de dispositivos diversos (é possível conectar dezenas deles) através de dois fios identificados como SDA (**S**erial **DA**ta) e SCL (**S**erial **CL**ock). Criado pela Philips, o padrão é muito utilizado em dispositivos como celulares para interligar componentes internos com boa performance e funcionalidade. Você pode encontrar maiores detalhes, por exemplo, no Wikipedia (*https://pt.wikipedia.org/wiki/I²C*). No entanto, utilizá-lo na prática é relativamente simples, devido à existência de farta documentação e diversas bibliotecas na internet.

Em termos comparativos, embora o SPI seja mais rápido e consuma menos energia que o i2C, seus dispositivos compatíveis são tipicamente mais caros. Além disto, o padrão exige mais conexões (quatro contra as duas do i2C). Isto torna interessante a escolha do i2C, muito embora o mais comum é adotar o padrão não por suas características, e sim infelizmente pela disponibilidade de periféricos compatíveis.

25.8 Conexões ao Arduíno Uno



A placa de protótipo Arduíno Uno foi projetada para simplificar o trabalho de estudantes e professores através de conexões simples, sem exigência de ferramentas como ferro de solda, ou instrumentos de medição.

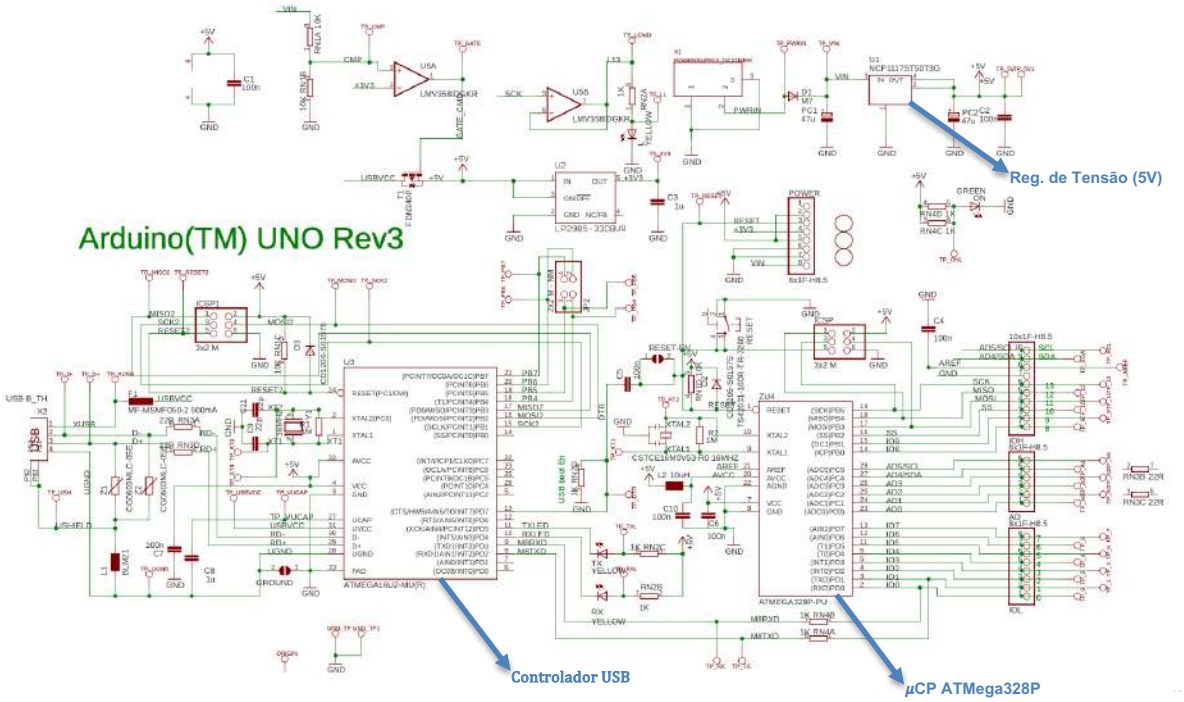
Para as conexões, o Arduíno oferece barras de terminais fêmea que permitem estabelecer conexões elétricas através de fios "*jumpers*", trazendo maior agilidade na montagem de protótipos e experimentos.

Na foto, vemos conexões feitas através de fios jumpers à barra de terminais de uma placa Arduíno Uno, neste caso utilizando algumas das portas digitais disponíveis.

25.9 Por dentro do Arduíno Uno R3

A placa de protótipo do Arduíno Uno R3 é constituída de diversos elementos. O diagrama esquemático abaixo representa os componentes da placa de protótipo e suas interligações. Este diagrama está sendo apresentado apenas como uma curiosidade, já que, assim como em um computador pessoal, não é necessário compreendê-lo para utilizá-lo em seus projetos.

Na figura, estão indicados 3 componentes muito importantes: o microcontrolador principal ATMega328P, o controlador USB e o regulador da fonte de alimentação de 5V.



26 Outras placas Arduíno

O projeto Arduíno resultou não só na placa Arduíno Uno, mas também em outros modelos com diferentes características. Alguns são muito populares, embora estejam descontinuados:

Arduíno Nano: a placa Nano, também é muito popular, e oferece basicamente os mesmos recursos do Arduíno Uno, porém ocupando um espaço bem menor. Por outro lado, se o objetivo for desenvolver protótipos, a placa Nano exige a soldagem dos seus pinos para conexão ao ProtoBoard. A figura, retirada de um anúncio do produto na Amazon, mostra as barras de pino antes da soldagem à placa. Ele utiliza outro layout de pinos, oferecendo 8 portas analógicas, porém apenas uma conexão com o GND.

Arduíno Mega: oferece um número bem maior de pinos de interface (54 digitais, com 15 PWM, e 16 analógicas), 4 interfaces UARTs, além de mais espaço de armazenamento para dados (8KB SRAM) e *sketches* (256KB). Utiliza o microcontrolador ATMega2560, e é fisicamente maior. Possui a mesma estrutura de pinos do Arduíno Uno, sendo inclusive compatível com boa parte dos *shields* desenvolvidas para o Uno.

Arduíno Lilypad: tem basicamente as mesmas características do Arduíno Uno, porém seu design foi criado com o foco em aplicações "vestíveis", ou seja, ele pode ser costurado em roupas e acessórios, por ser achatado e ocupar muito pouco espaço. Devido ao espaço reduzido, ele possui limitações de *hardware* (processador mais simples (ATMega328V), menos memória (SRAM de 1KB, Flash de 16KB) e clock mais baixo (8MHz). Além disto, não dispõe de interface USB. A placa pode ser alimentada com tensões entre 2,7 e 5,5 V, tornando-o mais flexível.

Os produtos descontinuados foram substituídos por novos, que tipicamente oferecem outras funcionalidades, mais performance e flexibilidade. Um bom exemplo é o aperfeiçoamento do clássico Arduíno Uno. Temos agora a versão 4, disponível em dois modelos:

Arduíno Uno R4 Minima: embora seja compatível em muitos pontos com a versão anterior (pinagem e tensão de alimentação, por exemplo), a nova versão do Arduíno Uno trouxe algumas melhorias:

- Adotou um novo microcontrolador, o RA4M1 da Renesas, de 32bits. Ele traz embutido um microprocessador ARM Cortex-M4 com *clock* de 48MHz (3 vezes maior), e uma unidade de cálculo de ponto flutuante;

- A memória SRAM aumentou para 32KB (16 vezes maior), a EEPROM para 8KB (8 vezes maior), e a memória flash para 256KB (também 8 vezes maior);

- Substituiu o regulador de tensão linear por uma fonte chaveada, suportando agora uma faixa maior de tensões de entrada (entre 6 e 24V), e uma carga máxima de até 1,2 A.

- Incorporou um RTC (Relógio de Tempo Real);

- Foi adicionado um conversor DAC de 12 bits;

- O ADC agora é de 14 bits, e não 10 como na versão 3;

- Incorporou um sensor de toque capacitivo;

- Suporta, além do UART, I2C e SPI disponíveis na versão 3, agora temos o protocolo CAN (através de um transceptor externo não incluído);

- A nova versão é compatível com o padrão HID, permitindo simular periféricos típicos de dispositivos computacionais, como teclado e mouse, por exemplo;

Uma outra mudança foi a conexão USB, que mudou da versão B para USB-C, e agora com a performance plena do padrão USB 2.0.

Arduíno Uno R4 Wi-Fi: esta versão tem as mesmas características da anterior, além dos seguintes novos recursos:

- Módulo ESP32-S3: este módulo oferece suporte a Bluetooth e Wi-Fi nativos;

- Matriz de LEDs 12x8: nesta versão foram instalados 96 LEDs na placa de protótipo formando uma matriz que pode ser utilizada para mostrar informações diversas;

- Segunda porta I2C, disponível através de conector no padrão Qwiic;

- Conector específico para alimentação do RTC.

27 Outras placas de protótipo

Além dos produtos derivados do projeto Arduíno, encontra-se no mercado outras soluções de placas de protótipo com diferentes recursos, baseadas em outros microcontroladores.

A depender de suas demandas técnicas e verba disponível você pode escolher entre diversas opções. Eis algumas:

NodeMCU: é uma linha baseada nos microcontroladores da linha ESP8266. Oferece placas de diferentes tamanhos e recursos. Destaca-se pela excelente relação custo/benefício;

Teensy 4: a linha é baseada no microcontrolador ARM Cortex-M7 de 600MHz. Destaca-se pela performance acima da média das concorrentes;

Placas baseadas nos microcontroladores MSP430: esta linha da Texas Instruments, reconhecido fabricante de componentes eletrônicos, destaca-se pelo consumo reduzido, o que pode ser muito importante em alguns projetos;

Placas baseadas no STM32: oferecem um conjunto de recursos, performance e compatibilidade muito interessante, sendo muito versátil para os mais diversos projetos.

Entre os recursos comuns a estas placas, que não são normalmente encontradas na linha Arduíno, exceto para alguns modelos mais recentes, são as interfaces Wi-Fi e Bluetooth integradas, além de outras diferenças específicas, como a alimentação com tensão de 3,3V, entre outras.

28 Programando um Microcontrolador

Projetos de computação física não envolvem apenas o *hardware* das interfaces, sensores e microcontroladores, mas também a programação destes componentes para que eles executem a aplicação proposta pelo usuário. Os programas escritos para estas aplicações são chamados de "*sketches*".

Para programar a placa Arduíno Uno, por exemplo, utilizamos a IDE gratuita disponível no site do projeto. A IDE possui versões para diversas plataformas para computadores pessoais (Windows, MacOS, Linux), embora não haja suporte oficial para uso em dispositivos móveis baseados no iOS ou Android, salvo via WEB (comentaremos mais sobre isto abaixo). A linguagem de programação padrão é o C++.

A IDE pode ser utilizada para programar qualquer uma das placas de protótipo do projeto Arduíno, e até mesmo placas de outros fabricantes; basta configurar adequadamente a porta de conexão, e o tipo de placa.

A figura mostra a versão 1.8.19 da IDE quando é aberta pela primeira vez, ainda sem nenhum *sketch* digitado ou recuperado. Recomendamos esta versão pelo fato da mesma ser estável, e ter sido testada não só em placas de protótipo da linha Arduíno, mas também de outras fabricantes.

A fundação Arduíno oferece versões mais atuais, com mais recursos, e até mesmo uma versão WEB, porém você poderá encontrar problemas em alguns casos, ao menos até o momento em que esta versão deste documento foi produzida. Existe também a possibilidade de adotar outras IDEs, e até mesmo outras linguagens de programação, como o μPython, por exemplo. Neste curso não vamos abordar estas opções alternativas. ***

28.1 Linguagem de Programação e Seções obrigatórias

Conforme informamos anteriormente, os *sketches* normalmente são escritos na linguagem de programação C++. No caso da IDE Arduíno, eles estão divididos em duas partes: a seção ***setup()***, que contém os comandos que devem ser executados apenas uma vez, e a seção ***loop()***, que contém os comandos que serão executados continuamente (em *loop*, como o próprio nome da seção diz).

Embora normalmente a sessão *setup()* seja utilizada para comandos de configuração inicial do microcontrolador e suas interfaces internas e externas, não há restrição ao tipo de comando que pode ser executado. O conceito básico é de que os comandos dentro desta seção serão executados apenas uma vez.

Se não houver nenhum trecho do código que precise ser repetido, a seção *loop()* pode inclusive ser deixada vazia. O mesmo vale para a sessão *setup()*, que pode ser deixada vazia se não houver comando a ser executado apenas uma vez. É recomendável, no entanto, que sessões, mesmo vazias, existam dentro do código.

28.2 Carga e Execução do *Sketch*

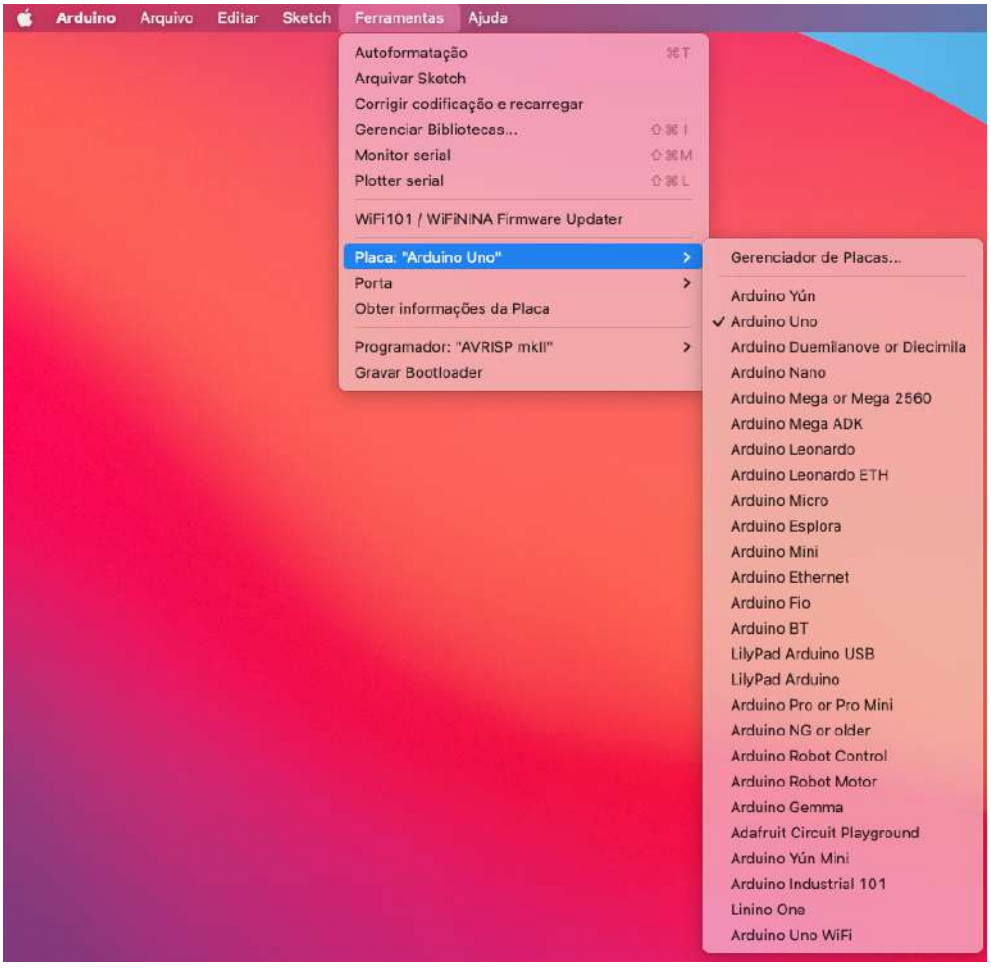
Placas de protótipo normalmente são programadas através de um compilador cruzado, ou seja, você utiliza o seu computador pessoal para criar, editar e compilar o seu sketch, e depois transfere o mesmo para a placa de protótipo através de uma interface de comunicação (ex.Porta USB). Isto é semelhante ao procedimento adotado para programar um celular, por exemplo.

Uma vez concluída a edição do *sketch*, para transferi-lo para a placa de protótipo, esta deve ser interligada ao computador através de uma interface USB. Algumas versões de placas do projeto Arduíno, e também de outros fabricantes, podem exigir interfaces específicas para conexão ao computador, por não ofertar interfaces USB embutidas.

A conexão, no entanto, não é apenas "física". A IDE precisa ser configurada para acesso à porta USB escolhida conforme mostra a figura a seguir.

Outra configuração essencial é a seleção da placa conectada (neste caso um Arduíno Uno), o que também é feito na IDE, conforme mostram as figuras a seguir. Na figura a seguir, veja que a IDE oferece suporte a diversos modelos de placa de protótipo da fundação Arduíno (o Nano, a Mega e a LilyPad aparecem como opções). A primeira opção do menu (Gerenciador de Placas) permite instalar os *drivers* de suporte para outros modelos de placa de protótipo, inclusive dispositivos não-Arduíno.

Ao selecionar uma determinada placa de protótipo, a IDE será capaz de preparar um código compilado compatível com o *hardware* da placa de protótipo conectada, considerando o microcontrolador instalado, capacidade de memória, interfaces e periféricos.



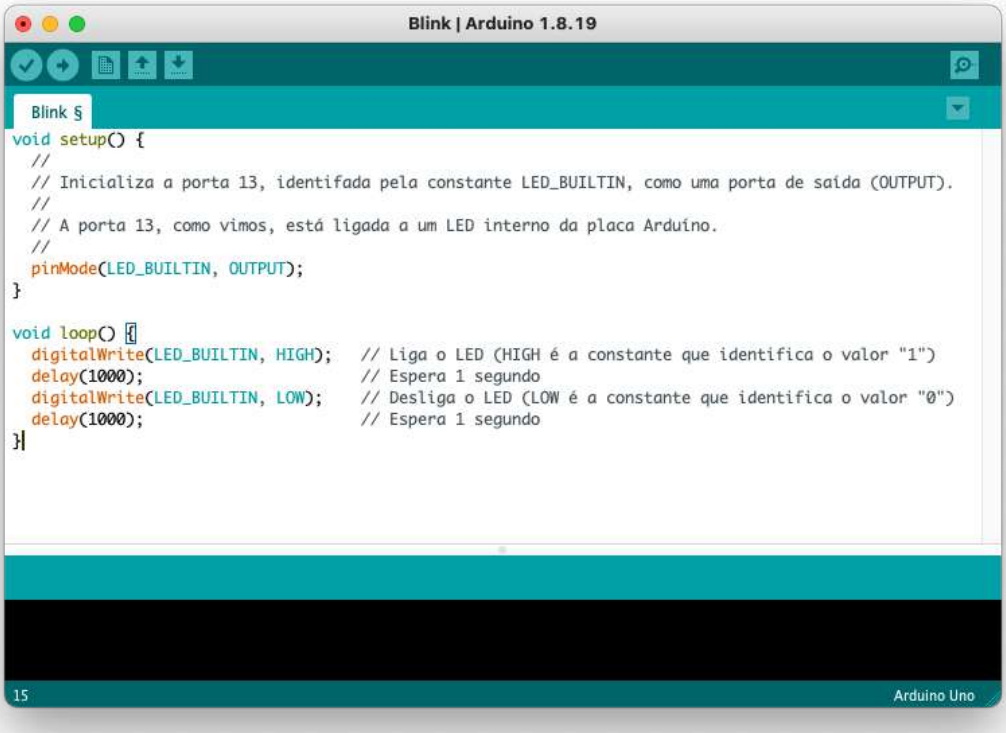
Após esta configuração, um teste indicado é clicar na opção "Obter informações da Placa" do primeiro menu. Se a placa for reconhecida, uma mensagem deve ser apresentada. Se nenhuma mensagem aparecer, provavelmente você encontrará problemas quando precisar se comunicar com a placa, como por exemplo, para transferir o seu *sketch* para execução. Algumas vezes a mensagem apresentada não indica o modelo de placa, ou informa que se trata de uma placa "desconhecida". Mensagens como estas normalmente não afetam a operação do ambiente.

Feito isto, basta um simples "click" no botão de transferência da IDE, e o *sketch*, após compilado, será transferido para a placa, e começará a ser executado imediatamente.

28.3 Um exemplo de *sketch*

Toda linguagem de programação tem, em sua literatura, um exemplo clássico de código: é o "Hello, World!", um programa cujo objetivo é simplesmente escrever a frase na tela do dispositivo.

Na Computação Física, especialmente no Arduíno Uno, o substituto do "Hello, World!" é o "Blink". Este é um *sketch* que simplesmente faz o LED interno da placa piscar (*blink*). Vamos mostrar este exemplo na prática. A figura a seguir mostra a tela do IDE já com o código devidamente digitado.



Para analisar o *sketch*, vamos verificar as sessões do código e alguns dos comandos. A sintaxe deve parecer familiar para programadores C, C++ e Java.

Seção *setup()*: a configuração inicial define que o pino 13, que está interligado ao LED interno da placa, será utilizado como uma saída ("OUTPUT"), já que desejamos carregar valores 0 e 1 alternadamente na mesma, visando fazer o LED piscar. Eis alguns detalhes da instrução executada nesta seção:

pinmode() – função que permite escolher o modo de operação de uma determinada porta;

LED_BUILTIN – constante pré-definida e igual ao número inteiro 13. Você pode substituir pelo número 13 e obterá os mesmos resultados, mas usar a constante simplifica a análise e portabilidade do código para outras placas de protótipo no futuro.

OUTPUT – constante pré-definida que determina a operação da porta específica como uma saída.

Seção *loop()*: a sessão que será executada continuamente repete indefinidamente dois conjuntos das instruções abaixo:

digitalWrite() – função que escreve um determinado valor em uma porta definida previamente como OUTPUT;

HIGH – constante pré-definida e igual ao valor lógico "1".

LOW – constante pré-definida e igual ao valor lógico "0".

delay(1000) – função que determina que o microcontrolador espere por 1000 milisegundos (que é igual a 1 segundo) antes de continuar a execução.

Com a troca dos valores escritos na porta 13 sendo repetida indefinidamente, teremos o efeito desejado, ou seja, o LED instalado na placa vai piscar 1 vez a cada 2 segundos. Você pode, após confirmar a execução, testar a troca dos valores passados como parâmetros para a função delay() para verificar o aumento ou redução do ritmo das piscadas, ou do tempo em que o LED fica aceso, ou apagado.

28.4 Estrutura da linguagem e sintaxe

Como o Arduíno IDE é baseado na linguagem C++, ele é compatível com as estruturas de teste, repetição, operadores e demais componentes da linguagem. Destas, a maioria são as mesmas estruturas utilizadas tanto pela linguagem C como pelo Java. Isto deve facilitar o seu aprendizado caso tenha experiência em alguma destas linguagens. O estudo específico destas estruturas foge do escopo desta disciplina.

Estrutura Básica

loop() setup()

(específicos da IDE Arduíno)

Estrutura Avançada

#define e **#include** são comandos do pré-processador, recurso específico da linguagem C, inexistente no Java.

#define #include // /* ... */ { ... }

Controle

if ... else switch ... case do ... while while for

Operadores de Comparação

= = ! = < > <= >=

Operadores Booleanos

! && ||

Aos itens básicos da linguagem C foram incluídas algumas funções específicas da computação física, basicamente para lidar com entradas, saídas, interrupções, gerência de tempo etc. Todas as funções encontram-se devidamente documentadas no *site* do projeto Arduíno, e também em diversos *sites* na internet.

Digitais

digitalRead() digitalWrite() pinMode()

Analógicas

analogRead() analogReference() analogWrite()

Comunicação

Serial.? Stream.? Keyboard.? Mouse.?

Tempo

delay() delayMicroseconds() micros() millis()

Avançadas

tone() shiftIn() shiftOut() pulseIn() pulseInLong() noTone()

28.5 Algumas limitações importantes

Como qualquer placa de protótipo, o Arduíno Uno não dispõe de Sistema Operacional instalado. Sendo assim, diversos recursos que podem, a primeira vista, parecer fundamentais para um usuário de computador não estarão disponíveis. Alguns exemplos são o suporte a aplicações multitarefa; a conexão direta a teclado, mouse, monitor de vídeo; as conexões a rede ethernet, wi-fi e bluetooth; a gerência e armazenamento de arquivos.

Outra limitação importante é de *hardware*. Placas de protótipos não são projetadas para ofertar grande desempenho, ou grande capacidade de memória. Sendo assim, embora os recursos sejam mais do que suficientes para praticamente todas as aplicações de computação física (para as quais os dispositivos foram projetados), podem faltar recursos para funcionalidades avançadas, como processamento gráfico, algoritmos matemáticos mais complexos, aprendizado de máquina etc. Por outro lado, uma placa de protótipo oferece recursos de computação física, como o acesso direto a múltiplos dispositivos externos de entrada e saída, que são inexistentes em um computador pessoal, mesmo nos mais sofisticados.

No caso do desenvolvimento, o IDE padrão é limitado nos recursos de *debug*, interface com dispositivos e periféricos, entre outras limitações.

Nada disto, no entanto, vai atrapalhar a sua criatividade, pode estar certo. As placas de protótipo, tal como o Arduíno Uno, oferecem recursos mais do que suficientes para projetos, inclusive alguns relativamente complexos.

29 Simulando projetos com placas de protótipo

Nem sempre temos acesso a um kit físico do Arduíno Uno ou de outra placa de protótipo. Isso, no entanto não impede que você desenvolva projetos, teste *sketches* e até alguns circuitos lógicos. Para isto você pode utilizar um simulador.

No programa gratuito TinkerCAD, da AutoDesk, que é executado em nuvem, você pode encontrar dezenas de exemplos de projeto já prontos, tanto de *hardware* quanto de *software*. É bastante recomendável criar uma conta no simulador, e analisar os projetos disponíveis.

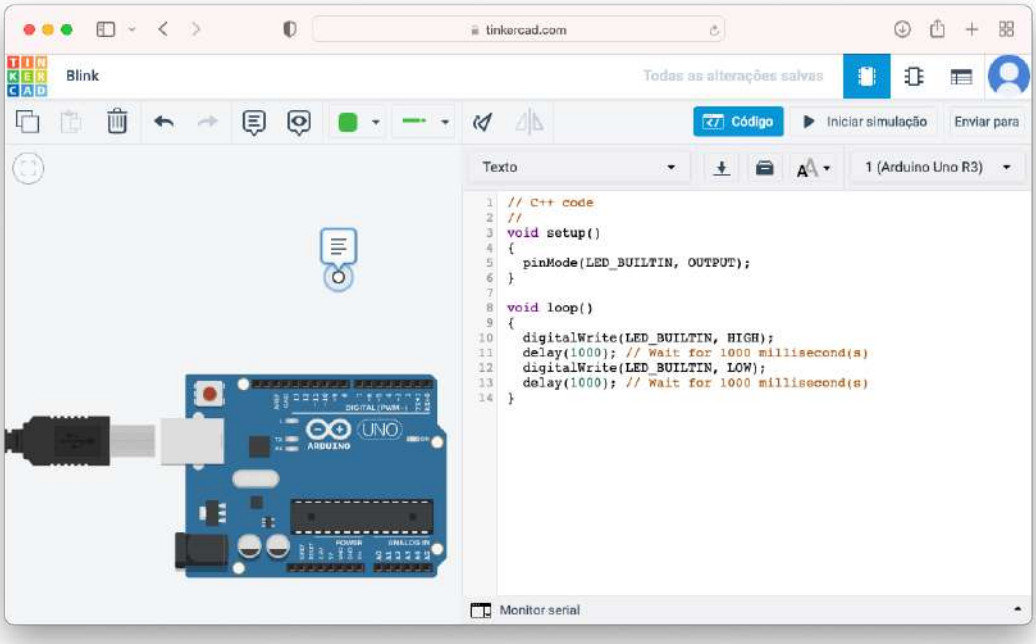
Ao carregar a página (https://www.tinkercad.com) após criar a sua conta, selecione "Projetos", depois "+ Criar", "Circuito". Na tela do projeto, selecione os componentes do lado direito e arraste para o projeto. Depois basta interligar os componentes, e quando tudo estiver pronto, basta pedir para "Iniciar simulação" e você verá o seu projeto funcionando.

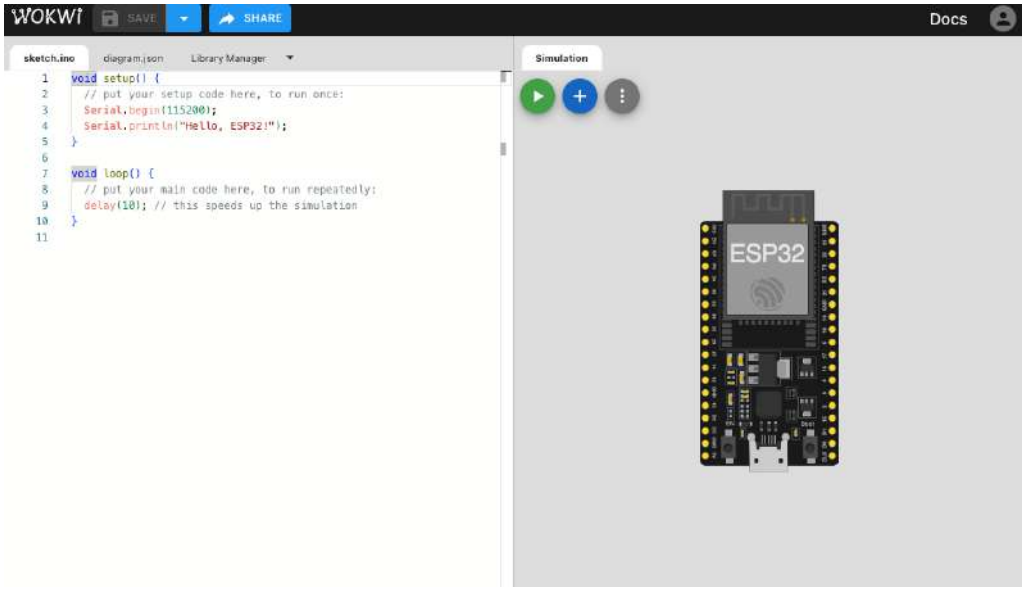
Se precisar de ajuda, o vídeo http://y2u.be/qDjW529R7BE, disponível no meu canal, mostra o processo para acessar a página.

Entre os componentes, você encontrará a placa de protótipo Arduíno R3, e diversos acessórios, como sensores, atuadores, *displays* etc. Além disto, estão disponíveis equipamentos de laboratório, como fonte de alimentação, multímetro, gerador de funções e até um osciloscópio. A brincadeira pode ser séria!

Quanto ao *software*, você pode programar o Arduíno de forma bem similar ao mundo real. Ao carregar o Arduíno Uno R3 no simulador, ele já vem com o código do Blink como exemplo, como você pode ver na figura a seguir.

Outra opção bastante interessante de simulador é o Wokwi (https://wokwi.com), que oferece inclusive outras placas de protótipo como a ESP32, STM32 ou o Raspberry Pi Pico.


tinkercad.com
Blink
Todas as alterações salvas
Código
Iniciar simulação
Enviar para
Texto
1 (Arduino Uno R3)
// C++ code
//
void setup()
{
pinMode(LED_BUILTIN, OUTPUT);
}
void loop()
{
digitalWrite(LED_BUILTIN, HIGH);
delay(1000); // Wait for 1000 millisecond(s)
digitalWrite(LED_BUILTIN, LOW);
delay(1000); // Wait for 1000 millisecond(s)
}
Monitor serial


WOKWI
SAVE
SHARE
Docs
sketch.ino
diagram.json
Library Manager
Simulation
void setup() {
// put your setup code here, to run once:
Serial.begin(115200);
Serial.println("Hello, ESP32!");
}
void loop() {
// put your main code here, to run repeatedly:
delay(10); // this speeds up the simulation
}
ESP32

30 REFERÊNCIAS

SILBERCHATZ, Abraham, GALVIN, Peter e GAGNE, Greg – Fundamentos de Sistemas Operacionais 9ª Edição, LTC [2015]

STALLINGS, William - Arquitetura e Organização de Computadores 8ªEdição, PEARSON [2010]

WIKIPEDIA MISD - https://en.wikipedia.org/wiki/Multiple_instruction,_single_data, link consultado em 30-07-2024.