Capítulo 1

Organização de um Computador

It is possible to invent a single machine which can be used to compute any computable sequence.

Alan Turing

Iniciemos com um exame, necessariamente superficial, da organização de um computador disponível em lojas de varejo em 2020[1]. A Figura 1.1 mostra um diagrama com os componentes principais no nosso computador. Vejamos a função de cada componente, e então um exemplo da operação conjunta de todos eles.

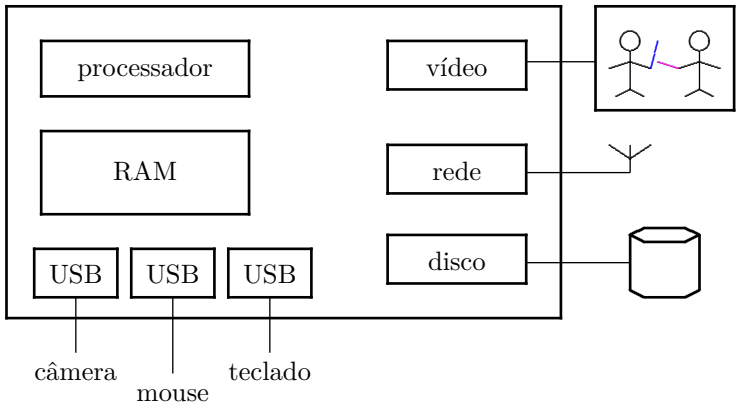

Figura 1.1: Organização de um computador pessoal.

Um pouco de notação Empregamos a notação de intervalos para denotar subconjuntos dos Inteiros, dos Naturais ou dos Reais. Um colchete fechado – '[' ou ']' – indica que o elemento adjacente ao colchete pertence ao conjunto, ou que ele faz parte do intervalo; por exemplo, o intervalo $[3, 7] \in \mathbb{N}$ inclui os Inteiros $3, 4, 5, 6, 7$. Um parêntese – '(' ou ')' – indica que o elemento adjacente *não* pertence ao intervalo; por exemplo, o intervalo $(0, 1] \in \mathbb{R}$ exclui o zero mas inclui todos os reais maiores que zero e menores ou iguais a 1.

Um *byte* é um octeto de dígitos binários e armazena um número natural no intervalo $[0, 255]$, ou um inteiro no intervalo $[-128, 127]$. Os *números mágicos* são $255 = (2^8 - 1)$, $-128 = -(2^7)$ e $+127 = (2^7 - 1)$. Essa tal *magia* é desmistificada no Capítulo 2.

[1] © Roberto André Hexsel, 2020. Versão de 4 de novembro de 2020.

Processador O *processador* é o componente principal de um computador pois é este quem interpreta e executa as *instruções* do programa que está a executar. As instruções representam comandos simples que o processador deve executar, tal como "adicione os conteúdos de x e y e armazene a soma em z", ou "armazene o valor de k na posição p da memória". Para consumo humano estas duas instruções são representadas como

```
z ← x + y
```

e

```
M(p) ← k.
```

No programa que o processador executa estas instruções são representadas por sequências de 32 dígitos binários (bits) tais como

```
11001010111100101110011010010001.
```

Evidentemente, a primeira representação é mais cômoda do que a segunda. A flecha que aponta para a esquerda representa a *atribuição* de um valor (`x+y`) a uma unidade de armazenamento (`z`), e o novo valor sobrescreve o conteúdo daquela unidade de armazenamento, que fora computado anteriormente.

Na disciplina Circuitos Digitais, estudamos a programação e a organização de um processador simples, que é descrito no Capítulo 9. Na disciplina de Projetos Digitais (Capítulos 10 e 11) veremos o projeto e a implementação de um processador realista. O projeto de computadores é objeto da disciplina Arquitetura de Computadores.

RAM Em Português, a memória RAM se chama *memória de escrita e de leitura*, embora o autor prefira a abreviatura do nome em Inglês: *Random Access Memory*. A razão para o *random* é histórica – nas décadas de 1940-50 a memória disponível era *sequencial* e para acessar a posição 19 da memória era necessário iniciar a busca pela posição 0, depois 1, 2, 3, ... 19. A possibilidade de acessos a quaisquer endereços, aleatoriamente, foi um grande progresso.

Do ponto de vista lógico, a RAM é uma função bijetora, os elementos do domínio são chamados de *endereços* e os elementos da imagem são o *conteúdo* armazenado nos respectivos endereços. Se a unidade de acesso é um byte (oito bits), então a memória é definida pela função

$$RAM : [0, C) \mapsto [0, 255]$$

sendo C a capacidade da memória, que é o número de bytes endereçáveis, e a cada endereço corresponde um valor representável em 8 bits, que é um número natural na faixa de 0 a 255.

Quando a memória é RAM, seus conteúdos podem ser alterados; quando a memória é ROM, ou *Read Only Memory*, seus conteúdos não podem ser alterados após o processo de fabricação ter sido concluído. Existem memórias que se comportam como ROM na maior parte do tempo, embora possibilitem que seu conteúdo seja alterado com o circuito em operação.

A organização interna de memórias ROM e RAM é introduzida na Seção 4.3.4. Voltaremos a esse assunto na disciplina de Projetos Digitais (Seções 5.5 e 5.6).

Interface de vídeo Talvez um nome mais apropriado do que *interface de vídeo* seja "tela de computador". Considere uma tela quadrada com 1024×1024 pontos, ou *pixeis*, com imagens em tons de cinza, de tal forma que cada pixel seja representado por um byte. Nesse caso, 0 representa um ponto branco, 255 representa um ponto preto, e 127 representa um ponto de tonalidade cinza-médio.

É necessária uma RAM com 1024×1024 bytes, ou 1 Mbytes, para armazenar uma cópia do que é exibido na tela. O controlador de vídeo varre esta memória e ilumina cada ponto da tela de acordo com o valor do byte que lhe corresponde. Para alterar a imagem, o processador altera o conteúdo da *memória de vídeo*.

Para uma tela colorida, são necessários 4 bytes por pixel, um para cada uma das cores fundamentais (vermelho, verde e azul) e mais um para a intensidade da iluminação do ponto. A representação mais rica envolve um aumento de um fator de quatro na capacidade de memória para manter a cópia da imagem em memória. Além de mais memória, o processamento de imagens coloridas é computacionalmente mais custoso do que para tons de cinza.

Interface de Rede Considere uma *interface de rede* que opere com mensagens com capacidade de 1 a 1024 bytes. No lado da recepção, as mensagens chegam a qualquer instante e por isso a interface deve conter memória para acomodar algumas mensagens, caso o processador não esteja disponível para tratá-las no instante em que são recebidas. No lado da transmissão, bastaria espaço para acomodar uma mensagem de tamanho máximo; por razões de eficiência, também aloca-se espaço para algumas mensagens no circuito de transmissão.

Como deve ser gerenciado o fluxo de mensagens para possibilitar a transferência de um arquivo grande? Como a ordem das mensagens é preservada? É desejável que ocorram várias transferências simultaneamente? Como os fluxos distintos são mantidos separados? Como os computadores ligados à Internet são endereçados? O que fazer quando da ocorrência de erros? Quais seriam os erros? Respostas precisas para estas perguntas serão obtidas ao cursar a(s) disciplina(s) de Redes de Computadores.

Disco Magnético como Memória Secundária A RAM é chamada de *memória primária* porque é essa memória que o processador acessa diretamente para executar programas e ler ou atualizar dados. Contudo, a capacidade da RAM é menor do que o conjunto de dados com que trabalhamos, e assim a *memória secundária* é empregada para armazenar arquivos tais como livros, filmes, etc. Em 2020, a memória secundária é tipicamente implementada com discos magnéticos. Como o nome indica, o 'disco' é coberto com óxido ferroso e porções diminutas da superfície do disco são magnetizadas para codificar 0s e 1s.

Cada superfície de um disco magnético é organizada em trilhas concêntricas, cada trilha dividida em setores. Um setor armazena de 1024 a 4096 bytes. Uma unidade de disco magnético pode ter mais de uma superfície. A posição de um determinado byte no disco é dada pela tripla $\langle \text{setor}, \text{trilha}, \text{superfície} \rangle$.

Por uma série de razões que serão discutidas mais adiante no curso, é conveniente trabalhar com um *disco lógico*, que é endereçado somente pelo número do bloco desejado. Em geral, um *bloco lógico* tem o mesmo tamanho de um setor, ou um *bloco físico*.

O controlador do disco faz a conversão de endereços físicos de 3 dimensões para endereços lógicos de uma dimensão

$$\langle \text{setor}, \text{trilha}, \text{superfície} \rangle \mapsto \langle \text{número do bloco} \rangle.$$

O endereço de um byte no disco lógico pode ser determinado pelas funções

$$\mathit{disco} : [0, N) \mapsto \mathit{bloco}, \quad \mathit{bloco} : [0, B) \mapsto [0, 255]$$

sendo N o número de blocos, e B o tamanho de um bloco.

Dispositivos de armazenamento com memória de estado sólido (*Solid State Disks* ou SSDs) estão se tornando populares pelo seu tempo de acesso, que é cerca de 100 vezes mais curto do que o dos discos magnéticos. É provável que em breve, os discos magnéticos sejam suplantados por SSDs. SSDs são acessados como discos magnéticos, e indexados pelo número do bloco. Em tese, estes dispositivos poderiam ser acessados como RAM, byte a byte ao invés de "bloco depois byte".

As memórias primária e secundária são estudadas nas disciplinas Software Básico e Sistemas Operacionais. A tradução de programas escritos em Pascal ou C para linguagem de máquina é estudada aqui, em Projetos Digitais, Software Básico e em Construção de Compiladores.

Exemplo 1.1 *Vejamos como é o fluxo dos bytes quando obtemos uma cópia de um arquivo a partir de um servidor de conteúdos da Internet. A Figura 1.2 mostra uma versão ligeiramente expandida do computador da Figura 1.1.*

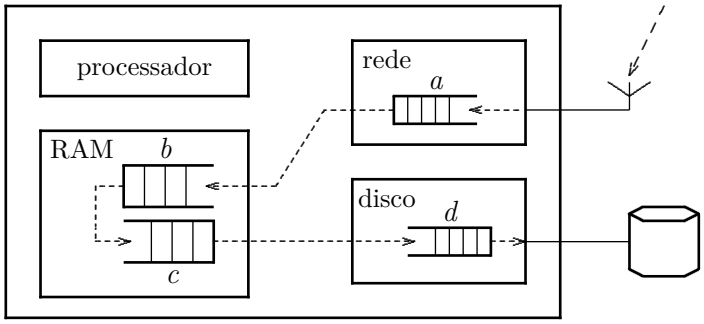

Figura 1.2: Fluxo de bytes entre interface de rede e memória secundária.

As mensagens são recebidas do servidor remoto em tempos imprevisíveis e indeterminados, e por isso emprega-se uma fila (a) para acomodar as diferenças na taxa em que as mensagens são recebidas, e a taxa com que elas são tratadas pelo processador. Esta fila tem a mesma função que a fila de um banco, aonde as pessoas (mensagens) esperam pelo atendimento por um dos caixas (processador). No caso da interface de rede, a fila é necessária para desacoplar as velocidades de chegada na antena e de atendimento pelo processador. A fila a é gerenciada pelo controlador da interface de rede, que é, ele próprio, um 'computador' dedicado a tratar da recepção e envio de mensagens através da interface de rede.

A fila b é usada para garantir que todas as mensagens que deveriam ter sido recebidas foram recebidas, e mais ainda, que a ordem dos bytes no arquivo original foi preservada durante a transmissão. Essa fila é gerenciada pelo sistema operacional (SO) do computador, e é acessada pelo processador para inserir novos elementos na cauda, e retirar os elementos mais antigos da cabeça da fila. Esta fila é parte do componente do SO que gerencia a transmissão e recepção de mensagens.

Uma vez que o SO verificou que o segmento do arquivo que está na fila b foi recebido corretamente, este segmento é removido da fila b e inserido na fila c. Esta fila é parte do componente do SO que gerencia o sistema de arquivos, e tem uma finalidade semelhante à da fila b, exceto que, ao invés de mensagens, os elementos desta fila são blocos lógicos que serão gravados no disco. Em geral, o tamanho dos blocos lógicos é distinto da capacidade máxima das mensagens, e é por isso que mensagens e blocos devem ser mantidos em filas separadas.

A fila d é gerenciada pelo controlador do disco, e seus elementos são blocos lógicos, esperando para ser gravados nos respectivos blocos físicos. O tempo de acesso de um disco magnético é longo e uma

escrita em disco demora o tempo equivalente à execução de 10^5 *a* 10^6 *instruções pelo processador. Para todos os fins práticos, o tempo de acesso aos blocos físicos é imprevisível, e a fila* d *acomoda as diferenças de velocidade com que blocos lógicos são nela inseridos pelo processador, e blocos físicos são removidos e gravados na superfície magnetizada pelo controlador do disco. Assim como o controlador da interface de rede, o controlador de disco é um 'computador' dedicado a controlar as transferências entre memória RAM e a superfície magnetizada.* ◁

Exercícios

Ex. 1.1 Quais as diferenças, do ponto de vista operacional, da aplicação da vacina tríplice em bebês e da vacina – quando esta ficar disponível – para a COVID-19?

Ex. 1.2 Considere um posto de saúde que atende a 100.000 pessoas, e que a vacina para COVID-19 está, finalmente, disponível. No posto existem quatro filas de aplicação de vacina, e cada aplicação demora 5 minutos. Supondo um horário de atendimento de oito horas, quantos dias são necessários para vacinar todas as 10^5 pessoas?

Ex. 1.3 Considerando os dados do Ex. 1.2, e que um lote de 5.000 doses é entregue ao posto a cada 3 dias, como isso altera o tempo de vacinação das 10^5 pessoas? Compensaria alterar o número de filas de aplicação? Se sim, como?

Ex. 1.4 Considerando os dados do Ex. 1.3, e que a fábrica de vacinas produz 1.000.000 de doses por semana, e supondo que todos os postos de saúde sejam como aquele do Ex. 1.2, quanto tempo levaria para vacinar todos os 220 milhões de brasileiros?

Ex. 1.5 Considerando os dados dos Ex. 1.2 a 1.4, compensaria alterar a logística de vacinação? Se sim, como?

Ex. 1.6 Podem ser necessárias duas doses para a imunização efetiva e eficaz. Como isso alteraria sua resposta ao Ex. 1.5?

Por mais estranho que possa parecer, estas questões são muito similares àquelas que devem ser respondidas por um projetista de computadores. As questões são abertas, e a resposta a uma questão posterior frequentemente impacta e altera as respostas para questões anteriores.

Referências Bibliográficas

[Ash08] Peter J Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, 3rd edition, 2008. ISBN 978-0-12-088785-9.

[BJ97] Gerrit A Blaauw and Frederick P Brooks Jr. *Computer Architecture: Concepts and Evolution*. Addison-Wesley, 1997. ISBN 0201105578.

[Bob87] Leonard S Bobrow. *Elementary Linear Circuit Analysis*. Holt, Rinehart & Winston, 1987. ISBN 0030072980.

[Bro04] Stuart Brorson. *Circuit simulation using gEDA and SPICE – HOWTO*, 2004. <http://www.brorson.com/gEDA/SPICE/intro.html>, 8/5/2012.

[Car03] Nicholas Carter. *Arquitetura de Computadores*. Coleção Schaum. Bookman, 2003. ISBN 853630250x.

[Cla80] Wesley A Clark. From electron mobility to logical structure: A view of integrated circuits. *ACM Computing Surveys*, 12(3):325–356, Set 1980.

[Fle97] William I Fletcher. *An Engineering Approach to Digital Design*. Prentice Hall, 1997. ISBN 9780132776998.

[Hex15] Roberto A Hexsel. cMIPS – a synthesizable VHDL model for the classical five stage pipeline. Repositório de *software*, Depto de Informática, UFPR, 2015. Disponível em <https://github.com/rhexsel/cmips>.

[Hex17] Roberto A Hexsel. Why it is so hard to write "system software"? To be submitted, 2017.

[HJS00] M D Hill, N P Jouppi, and G S Sohi. *Readings in Computer Architecture*. Morgan Kaufmann, 2000. ISBN 1558605398.

[HP90] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1st edition, 1990. ISBN 1558600698.

[HP12] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 5th edition, 2012.

[HU79] John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. ISBN 020102988X.

[Hwa93] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, 1993. ISBN 0070316228.

[JNW08] B L Jacob, S W Ng, and D T Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2008. ISBN 0123797513.

[Kat94] Randy H Katz. *Contemporary Logic Design*. Benjamin-Cummings, 1994. ISBN 0805327037.

[KB04] Randy H Katz and Gaetano Borriello. *Contemporary Logic Design*. Prentice Hall, 2004. ISBN 978-0201308570.

[KL96] Sung-Mo Kang and Yusuf Leblebici. *CMOS Digital Integrated Circuits: Analysis and Design*. McGraw-Hill, 1996. ISBN 0070380465.

[Koh78] Zvi Kohavi. *Switching and Finite Automata Theory*. Tata McGraw-Hill, 2nd edition, 1978. ISBN 0070993874.

[Kor01] Israel Koren. *Computer Arithmetic Algorithms*. A K Peters, 2nd edition, 2001. ISBN 1568811608.

[KP90] Al Kelley and Ira Pohl. *A Book on C, Programming in C*. Benjamin/Cummings, 2nd edition, 1990. ISBN 0805300600.

[KR88] Brian W Kernighan and Dennis M Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988. ISBN 9780131103628.

[Lam84] Butler W Lampson. Hints for computer system design. *IEEE Software*, pages 11–28, Jan 1984.

[Lip64] Seymour Lipschutz. *Set Theory and Related Topics*. Schaum Publishing, 1964.

[Man02] M Morris Mano. *Digital Design*. Prentice Hall, 3rd edition, 2002. ISBN 01306211218.

[MIP05a] MIPS. *MIPS32 Architecture for Programmers, Volume I: Introduction to the MIPS32 Architecture*, 2005.

[MIP05b] MIPS. *MIPS32 Architecture for Programmers, Volume II: The MIPS32 Instruction Set*, 2005.

[MIP05c] MIPS. *MIPS32 Architecture for Programmers, Volume III: The MIPS32 Privileged Resource Architecture*, 2005.

[MK00] M Morris Mano and Charles R Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, 2nd edition, 2000. ISBN 0130124680.

[Mou01] Arnaldo V Moura. *Especificações em Z*. Editora da Unicamp, 2001. ISBN 8526805754.

[PH94] David A Patterson and John L Hennessy. *Computer Organization & Design: The Hardware/Software Interface*. Morgan Kaufmann, 1994. ISBN 155860281X.

[PH14] David A Patterson and John L Hennessy. *Computer Organization & Design: The Hardware/Software Interface*. Morgan Kaufmann, 5th edition, 2014.

[PP03] Yale N Patt and Sanjay J Patel. *Introduction to Computing Systems: From Bits and Gates to C and Beyond*. McGraw-Hill, 2nd edition, 2003.

[RCN03] Jan M Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits – A Design Perspective*. Prentice Hall, 2nd edition, 2003. ISBN 0130909963.

[San90] Jeff W Sanders. Lectures on the foundations of hardware design. Lecture notes, Programming Research Group, Oxford University Computing Laboratory, 1990.

[Sco65] Ronald E Scott. *Elements of Linear Circuits*. Addison-Wesley, 1965. ISBN 0201068427.

[SCO96] The Santa Cruz Operation SCO. *System V Application Binary Interface – MIPS RISC Supplement*, 3rd edition, 1996.

[Spi89] J M Spivey. *The Z Notation*. Prentice Hall, 1989. ISBN 013983768X.

[SS90] Adel S Sedra and Kenneth C Smith. *Microeletronic Circuits*. Holt, Rinehart & Winston, 3rd edition, 1990. ISBN 003051648X.

[Swe07] Dominic Sweetman. *See MIPS Run – Linux*. Morgan Kaufmann, 2nd edition, 2007. ISBN 0120884216.

[Tau82] Herbert Taub. *Digital Circuits and Microprocessors*. McGraw-Hill, 1982. ISBN 0070629455.

[TS89] Herbert Taub and Donald Schilling. *Digital Integrated Electronics*. McGraw-Hill, 1989. ISBN 007Y857881.

[vN93] John von Neumann. First draft of a report on the EDVAC. *IEEE Ann History of Computing*, 15(4):27–75, Out 1993.

[WE82] Jim Welsh and John Elder. *Introduction to Pascal*. Prentice-Hall, 2nd edition, 1982. ISBN 0134915496.

[WH10] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 4th edition, 2010. ISBN 0321547748.

[Wol05] Wayne Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann, 2005. ISBN 0123694590.

Índice Remissivo