2.1.3 RISC *versus* CISC

Durante o final da década de 1970, houve experiências com instruções muito complexas que eram possibilitadas pelo interpretador. Os projetistas tentavam fechar a "lacuna semântica" entre o que as máquinas podiam fazer e o que as linguagens de programação de alto nível demandavam. Quase ninguém pensava em projetar máquinas mais simples, exatamente como agora não há muita pesquisa na área de projeto de planilhas, redes, servidores Web etc. menos poderosos (o que talvez seja lamentável).

Um grupo que se opôs à tendência e tentou incorporar algumas das ideias de Seymour Cray em um minicomputador de alto desempenho foi liderado por John Cocke na IBM. Esse trabalho resultou em um minicomputador denominado **801**. Embora a IBM nunca tenha lançado essa máquina no mercado e os resultados tenham sido publicados só muitos anos depois (Radin, 1982), a notícia vazou e outros começaram a investigar arquiteturas semelhantes.

Em 1980, um grupo em Berkeley, liderado por David Patterson e Carlo Séquin, começou a projetar chips para CPUs VLSI que não usavam interpretação (Patterson, 1985; Patterson e Séquin, 1982). Eles cunharam o termo **RISC** para esse conceito e deram ao seu chip de CPU o nome RISC I CPU, seguido logo depois pelo RISC II. Um pouco mais tarde, em 1981, do outro lado da baía de São Francisco, em Stanford, John Hennessy projetou e fabricou um chip um pouco diferente, que ele chamou de **MIPS** (Hennessy, 1984). Esses chips evoluíram para produtos de importância comercial, o SPARC e o MIPS, respectivamente.

Esses novos processadores tinham diferenças significativas em relação aos que havia no comércio naquela época. Uma vez que essas novas CPUs não eram compatíveis com os produtos existentes, seus projetistas tinham liberdade para escolher novos conjuntos de instruções que maximizassem o desempenho total do sistema. Embora a ênfase inicial estivesse dirigida a instruções simples, que podiam ser executadas rapidamente, logo se percebeu que projetar instruções que podiam ser **emitidas** (iniciadas) rapidamente era a chave do bom desempenho. Na verdade, o tempo que uma instrução demorava importava menos do que quantas delas podiam ser iniciadas por segundo.

Na época em que o projeto desses processadores simples estava no início, a característica que chamou a atenção de todos era o número relativamente pequeno de instruções disponíveis, em geral cerca de 50. Esse número era muito menor do que as 200 a 300 de computadores como o VAX da DEC e os grandes *mainframes* da IBM. De fato, o acrônimo RISC quer dizer **Reduced Instruction Set Computer** (computador com conjunto de instruções reduzido), em comparação com CISC, que significa **Complex Instruction Set Computer** (computador com conjunto de instruções complexo), uma referência nada sutil ao VAX que, na época, dominava os departamentos de ciência da computação das universidades. Hoje em dia, poucas pessoas acham que o tamanho do conjunto de instruções seja um assunto importante, mas o nome pegou.

Encurtando a história, seguiu-se uma grande guerra santa, com os defensores do RISC atacando a ordem estabelecida (VAX, Intel, grandes *mainframes* da IBM). Eles afirmavam que o melhor modo de projetar um computador era ter um pequeno número de instruções simples que executassem em um só ciclo do caminho de dados da Figura 2.2, ou seja, buscar dois registradores, combiná-los de algum modo (por exemplo, adicionando-os ou fazendo AND) e armazenar o resultado de volta em um registrador. O argumento desses pesquisadores era de que, mesmo que uma máquina RISC precisasse de quatro ou cinco instruções para fazer o que uma CISC fazia com uma só, se as instruções RISC fossem dez vezes mais rápidas (porque não eram interpretadas), o RISC vencia. Também vale a pena destacar que, naquele tempo, a velocidade de memórias principais tinha alcançado a velocidade de memórias de controle somente de leitura, de modo que a penalidade imposta pela interpretação tinha aumentado demais, o que favorecia muito as máquinas RISC.

Era de imaginar que, dadas as vantagens de desempenho da tecnologia RISC, as máquinas RISC (como a Sun UltraSPARC) passariam como rolo compressor sobre as máquinas CISC (tal como a Pentium da Intel) existentes no mercado. Nada disso aconteceu. Por quê?

Antes de tudo, há a questão da compatibilidade e dos bilhões de dólares que as empresas tinham investido em software para a linha Intel. Em segundo lugar, o que era surpreendente, a Intel conseguiu empregar as mesmas ideias mesmo em uma arquitetura CISC. A partir do 486, as CPUs da Intel contêm um núcleo RISC que executa as instruções mais simples (que normalmente são as mais comuns) em um único ciclo do caminho de dados, enquanto interpreta as mais complicadas no modo CISC de sempre. O resultado disso é que as instruções comuns são rápidas e as menos comuns são lentas. Mesmo que essa abordagem híbrida não seja tão rápida quanto um projeto RISC puro, ela resulta em desempenho global competitivo e ainda permite que softwares antigos sejam executados sem modificação.

2.1.4 Princípios de projeto para computadores modernos

Agora que já se passaram mais de duas décadas desde que as primeiras máquinas RISC foram lançadas, certos princípios de projeto passaram a ser aceitos como um bom modo de projetar computadores, dado o estado atual da tecnologia de hardware. Se ocorrer uma importante mudança na tecnologia (por exemplo, se, de repente, um novo processo de fabricação fizer o ciclo de memória ficar dez vezes mais rápido do que o tempo de ciclo da CPU), todas as apostas perdem. Assim, os projetistas de máquinas devem estar sempre de olho nas mudanças tecnológicas que possam afetar o equilíbrio entre os componentes.

Dito isso, há um conjunto de princípios de projeto, às vezes denominados **princípios de projeto RISC**, que os arquitetos de CPUs de uso geral se esforçam por seguir. Limitações externas, como a exigência de compatibilidade com alguma arquitetura existente, muitas vezes exigem uma solução de conciliação de tempos em tempos, mas esses princípios são metas que a maioria dos projetistas se esforça para cumprir. A seguir, discutiremos os principais.

• Todas as instruções são executadas diretamente por hardware

Todas as instruções comuns são executadas diretamente pelo hardware – não são interpretadas por microinstruções. Eliminar um nível de interpretação dá alta velocidade à maioria das instruções. No caso de computadores que executam conjuntos de instruções CISC, as instruções mais complexas podem ser subdivididas em partes separadas que então podem ser executadas como uma sequência de microinstruções. Essa etapa extra torna a máquina mais lenta, porém, para instruções que ocorrem com menos frequência, isso pode ser aceitável.

• Maximize a taxa de execução das instruções

Computadores modernos recorrem a muitos truques para maximizar seu desempenho, entre os quais o principal é tentar iniciar o máximo possível de instruções por segundo. Afinal, se você puder emitir 500 milhões de instruções por segundo, terá construído um processador de 500 MIPS, não importa quanto tempo elas realmente levem para ser concluídas. (**MIPS** quer dizer Milhões de Instruções Por Segundo. O processador MIPS recebeu esse nome como um trocadilho desse acrônimo. Oficialmente, ele significa **Microprocessor without Interlocked Pipeline Stages** – microprocessador sem estágios paralelos de interbloqueio.) Esse princípio sugere que o paralelismo pode desempenhar um importante papel na melhoria do desempenho, uma vez que emitir grandes quantidades de instruções lentas em curto intervalo de tempo só é possível se várias instruções puderem ser executadas ao mesmo tempo.

Embora as instruções sempre sejam encontradas na ordem do programa, nem sempre elas são executadas nessa mesma ordem (porque algum recurso necessário pode estar ocupado) e não precisam terminar na ordem do programa. É claro que, se a instrução 1 estabelece um registrador e a instrução 2 usa esse registrador, deve-se tomar muito cuidado para garantir que a instrução 2 não leia o registrador até que ele contenha o valor correto. Fazer isso funcionar direito requer muito controle, mas possibilita ganhos de desempenho por executar várias instruções ao mesmo tempo.

• Instruções devem ser fáceis de decodificar

Um limite crítico para a taxa de emissão de instruções é a decodificação de instruções individuais para determinar quais recursos elas necessitam. Qualquer coisa que possa ajudar nesse processo é útil. Isso inclui fazer instruções regulares, de comprimento fixo, com um pequeno número de campos. Quanto menor o número de formatos diferentes para as instruções, melhor.

• Somente LOAD e STORE devem referenciar a memória

Um dos modos mais simples de subdividir operações em etapas separadas é requerer que os operandos para a maioria das instruções venham de registradores da CPU e a eles retornem. A operação de movimentação de operandos da memória para registradores pode ser executada em instruções separadas. Uma vez que o acesso à

memória pode levar um longo tempo, e que o atraso é imprevisível, o melhor é sobrepor essas instruções a outras se elas nada fizerem exceto movimentar operandos entre registradores e memória. Essa observação significa que somente instruções LOAD e STORE devem referenciar a memória. Todas as outras devem operar apenas em registradores.

Providencie muitos registradores

Visto que o acesso à memória é relativamente lento, é preciso providenciar muitos registradores (no mínimo, 32) de modo que, assim que uma palavra for buscada, ela possa ser mantida em um registrador até não ser mais necessária. Esgotar os registradores e ter de descarregá-los de volta à memória só para ter de recarregá-los mais tarde é indesejável e deve ser evitado o máximo possível. A melhor maneira de conseguir isso é ter um número suficiente de registradores.