11

Arquiteturas RISC

11.1 INTRODUÇÃO

Desde os primeiros momentos da indústria de computadores que cientistas dos principais fabricantes, bem como pesquisadores de instituições acadêmicas e de centros de pesquisa, têm estudado, em maior ou menor intensidade (às vezes depende de demandas do mercado, outras vezes de interesses específicos de certas áreas de pesquisa), métodos e técnicas que possam aperfeiçoar o desempenho e a capacidade dos sitemas de computação. Tais pesquisas se desenvolvem tanto no que se refere à evolução das arquiteturas dos processadores e computadores quanto na parte de software, seja básico, como os sistemas operacionais, sejam aplicativos.

É interessante observar, ao longo de décadas de uso da computação, que o hardware em geral e os processadores e a memória em particular (nos interessa mais neste capítulo), têm evoluído mais rapidamente do que o software. Em outras palavras, os processadores têm crescido enormemente em termos de velocidade de operações (de poucos MHz em 1980 para alguns GHz em 2006), de tecnologia de processamento, como pipelining, superescalar, processamento paralelo, aumento de largura da palavra de 32 para 64 bits e outras e em capacidade de armazenamento (de alguns registradores internos para uma ou duas caches internas de dezenas e centenas de KB).

Por outro lado, o desenvolvimento e a manutenção de programas não têm evoluído em custo/benefício na mesma proporção, embora notáveis criações e lançamentos estejam surgindo a todo momento, mas sempre acarretando custos mais elevados (a falta de programadores qualificados é, sem dúvida, uma das razões), não só de desenvolvimento, mas também de manutenção, visto que os programas oferecidos pelos fabricantes de software estão quase sempre longe da ausência de falhas (*bugs*).

Em razão disso, foram surgindo, a partir do Fortran (a primeira linguagem de alto nível), novas e cada vez mais poderosas linguagens de programação, além de uma evolução dos conceitos de programação, mais sofisticados, como estruturação das linguagens, orientação a objetos, orientação a componentes e, mais recentemente, com sistemas multiagentes. Como os comandos e as estruturas existentes nessas linguagens são criados para atender ao modo de raciocinar do ser humano e à necessidade de reduzir o ciclo de vida dos sistemas de informações e aplicações, eles se afastam cada vez mais da simplicidade e do primitivismo do hardware, ou seja, das instruções que a máquina entende.

Em função desses fatores, constatou-se que estava acontecendo uma separação acentuada entre os comandos de linguagens de alto nível e as instruções de máquina decorrentes. Ou seja, um único comando de uma linguagem de alto nível significava muito, podendo gerar muitas instruções de máquina no código-objeto correspondente, dando bastante trabalho ao programa compilador (ver Apêndice C). Esta diferença de compreensão entre os comandos de linguagens de alto nível e de instruções de máquina ficou conhecida nas áreas de pesquisa como *semantic gap,* ou espaço semântico, e requereu decisões estratégicas de alguns fabricantes de processadores, em acordo com desenvolvedores de compiladores, entre os quais Intel, IBM e DEC (posteriormente adquirida pela Compaq, que, mais tarde, foi absorvida pela HP) e, posteriormente, AMD.

A decisão tomada por alguns dos fabricantes, inicialmente DEC e IBM, por serem fabricantes dos computadores de grande porte, predominantes na época, foi conduzindo, ao longo do tempo, ao crescimento da quantidade de instruções de máquina nos seus processadores. O objetivo era "fechar" o *gap* semântico, o que levou aqueles processadores a possuir uma considerável quantidade de instruções, com grande quantidade de modos de endereçamento e outras particularidades complexas.

Em determinada época (início da década de 1980), este tipo de arquitetura foi caracterizada como sendo de um determinado tipo chamado por alguns pesquisadores de CISC – *Complex Instruction Set Computers* ou **processadores com conjunto de instruções complexo**.

Em alguns processadores, a quantidade ultrapassava 300 instruções, como também a quantidade de modos de endereçamento, como no VAX, era enorme. A idéia dos fabricantes desses processadores era, como já mencionado, facilitar a construção e o serviço dos compiladores, procurando reduzir o referido gap semântico, embora isso não ocorresse na maioria dos casos, visto que os compiladores desprezavam a maior parte das instruções e modos de endereçamento.

Além dos processadores de grande (IBM) e médio (DEC/VAX) portes, também os fabricantes de microprocessadores (Intel e posteriormente AMD) procuraram o mesmo caminho, e a arquitetura x86 pôde ser considerada também como tendo as características identificadas como CISC.

Nesse ponto, parece interessante observar que os termos CISC e RISC surgiram quando se procurava encontrar arquiteturas de processadores mais rápidos e baratos. Provavelmente, foi o projeto do processador RISC 1, desenvolvido na Universidade da Califórnia, Berkeley, que adotou pela primeira vez esse nome, bem como os artigos decorrentes, escritos na época, denominaram CISC a arquitetura até então vigente com a Intel, DEC, IBM, etc. Mais adiante, veremos que os conceitos de um processador simples e rápido (objetivo de completar uma instrução por ciclo de máquina) são anteriores ao RISC 1.

Parece, então, que o termo CISC foi inserido na literatura e na discussão universitária como uma crítica ao modo de estrutura de alguns processadores e que, em conseqüência, os inventores do nome (CISC) criaram também a sigla RISC em contrapartida àquelas estruturas.

Ainda visando ao aperfeiçoamento dos projetos, não só de desenvolvimento das linguagens de alto nível mas também de seu comportamento, tanto estático, quando se transformam em instruções de máquina, quanto dinâmico em tempo de execução, várias pesquisas foram realizadas, desde a década de 1960, mas também em 1970 e 1980, sobre o comportamento dos programas, obtendo-se conclusões interessantes sobre o assunto. A intenção era também tentar encontrar outros modos de reduzir o gap semântico, em vez de somente criar instruções de máquina mais complexas e em maior quantidade.

A literatura possui várias dessas pesquisas, entre as quais pode-se citar a primeira delas, publicada em 1971 por Donald Knuth [KNUTH71], sobre o comportamento de vários programas codificados na linguagem Fortran, cujo resumo é mostrado na Fig. 11.1.

Comando	Média de ocorrência (em percentagem)
ASSIGN	47
IF	23
CALL	15
LOOP	6
GOTO	3
Outros	7

Figura 11.1 Percentagem da freqüência de ocorrência de Comandos Fortran em vários programas (KNUTH71).

Daquele estudo, pode-se observar que grande parte das instruções existentes, em média, nos programas resulta de instruções de atribuição – "assign" (ou movimento de dados), ou seja, transferência de dados de um local para outro, e sendo de uma memória implica a execução de uma demorada operação de leitura e/ou escrita. Verifica-se , ainda, o fato interessante de que apenas 7% são de "outras" intruções, nas quais se incluem as que realizam operações matemáticas e mais complexas.

E, mais ainda, se o desejo é aumentar o desempenho do sistema de computação, deve-se, então, procurar otimizar o emprego das instruções que consomem mais tempo de execução (como os "assign"), em vez de se ocupar com instruções mais complexas (próximas de um comando de linguagem de alto nível), mas que raramente são usadas.

Posteriormente, no início da década de 1980 surgiram várias pesquisas sobre o mesmo assunto, com o propósito específico de encontrar uma arquitetura de processadores que obtivesse os resultados desejados (reduzir o gap semântico entre os comandos de linguagens de alto nível e as instruções de máquina correspondentes), mas reduzindo os tempos de processamento e a complexidade do hardware.

Entre as pesquisas realizadas e os desenvolvimentos surgidos na época, pode-se citar o de David Patterson, que publicou em 1982 [PATT82], juntamente com Carlo Séquin, ambos da Universidade da Califórnia, em Berkeley, estudo mostrando o desempenho, os parâmetros e elementos de linguagens de alto nível quando compiladas e executadas. O artigo descrevia uma nova arquitetura para um processador, que visava solucionar os problemas de desempenho e custo existentes nas arquiteturas complexas (CISC) vigentes; esta arquitetura foi chamada de RISC, porque criava um processador com pequeno (reduzido) conjunto de instruções, conforme mencionado acima.

Além das pesquisas de Patterson, também surgiram resultados semelhantes de outros cientistas, como John Hennessy, da Universidade de Stanford; David Ditzel, dos laboratórios Bell, que publicou trabalhos, sozinho [DITZ80] e com Patterson [PATT80], sobre medidas de desempenho em programas. Um pouco antes, surgiram trabalhos de A. Lunde [LUND77] e B. A. Wichmann [WICH76], bem como o de Andrew Tanembaum [TANE78].

O trabalho mais citado como referência para avaliação do comportamento dos programas de alto nível e seus códigos-objeto gerados, e quando em execução (desempenho dinâmico), é um artigo de Patterson e Séquin [PATT82], no qual eles apresentam os resultados da compilação e da execução de oito programas diferentes, quatro em linguagem Pascal e quatro em linguagem C.

O propósito dos autores era o de verificar que tipo de estrutura de programação era usado mais freqüentemente e quais dessas estruturas redundavam, após a compilação e na fase de execução, em mais tempo de processamento, já como instruções de máquina. Em primeiro lugar, foram analisadas a existência e ocorrência dinâmica dos tipos de dados nos referidos programas, conforme resumo mostrado na tabela da Fig. 11.2.

Tipo de dado	Média em Pascal	Média em C	Média total
Constante inteira	(14,18,11,20) = 16%	(25,11,29,28) = 23%	20%
Variável escalar	58%	53%	55%
Array/Estruturas	26%	24%	25%

Obervação: 1. Utilizados quatro programas em Pascal (um compilador, uma macro em programa de design, um programa que analisa dois arquivos e um programa de impressão) e quatro programas em C (um compilador; um programa formatador de texto; um programa que realiza classificação de itens – sort e um programa de plotagem).
2. A partir da esquerda, a primeira coluna indica o tipo de dado (variável) existente nos programas; a segunda coluna mostra a média das medidas para os programas em Pascal; a terceira coluna mostra a média das medidas para os programas em C, e a última coluna, da direita, apresenta a média total das percentagens dos oito programas.

Figura 11.2 Tabela apresentando uma média da percentagem de ocorrência dinâmica de operandos em programas (adaptado de PATT82).

Da tabela, pode-se verificar que as constantes aparecem praticamente com a mesma freqüência que os arrays ou estruturas e, juntando com os resultados de [TANE78], verifica-se que mais de 80% dos escalares são variáveis locais, enquanto mais de 90% dos arrays e estruturas são variáveis globais.

Em seguida, o estudo passou a verificar a freqüência dinâmica de ocorrência de comandos de alto nível (como assign, if, call, etc.) dos mesmos oito programas. Combinando os resultados encontrados com o código de máquina gerado por cada comando, segundo dados obtidos de estudos internos em Berkeley por W. Wulf, obtiveram-se os elementos resumidos em uma única tabela, mostrada na Fig. 11.3.

Comando	Ocorrência		Peso nas instr. máquina		Peso em ref. à MP	
	Pascal	C	Pascal	C	Pascal	C
Assign	45%	38%	23%	13%	14%	15%
Loop	5%	3%	42%	32%	33%	26%
Call	15%	12%	31%	33%	44%	45%
LF	29%	43%	11%	21%	7%	13%
Goto	–	3%	–	–	–	–
Outros	6%	1%	3%	1%	2%	1%

Figura 11.3 Quadro comparativo da freqüência de ocorrência de certos comandos de linguagens de alto nível na execução de programas.

Em resumo, pode-se observar que:

a) Parece, da Fig. 11.3, que os comandos de chamada e retorno de rotinas (Call) são os que consomem mais tempo do processador em típicos códigos de alto nível. Os dados mostrados na tabela da Fig. 11.2, sobre dados, indicam a importância do emprego de variáveis locais e constantes.

b) O corpo básico de um programa (o "grosso" do programa, como se costuma falar informalmente) é simples, em termos de suas instruções de máquina, independentemente da possibilidade de serem criados programas mais complexos. Isso significa que esta complexidade adicional (até para reduzir o gap semântico) implica pouco significado no cômputo geral da execução do código de máquina.

E, mais ainda, as permanentes análises que são efetuadas sobre o código compilado por máquinas de arquitetura complexa (CISC) mostram que os compiladores não são tão espertos quanto os programadores assembly (hoje raros, mas nas décadas de 1980 e 1990 eram muitos) em buscar instruções de máquina sofisticadas. Eles usam muito pouco da grande quantidade de instruções e de modos de endereçamento disponíveis, pois parece ser bastante difícil ao programa compilador analisar o código de alto nível em suficiente detalhe para identificar que elementos requerem as instruções mais complexas que facilitariam a conversão. A razão mais aceitável parece ser a diferença natural entre o conceito e as estruturas das linguagens de alto nível e os das linguagens assembly.

Em resumo, o encontro de uma solução para se obter maior rapidez na operação dos processadores parece estar mais na definição de arquiteturas simples (que operam, por isso, mais rapidamente) do que as que são mais complexas (para se reduzir o gap semântico), que consomem mais tempo de operação.

Consideremos, por exemplo, em uma máquina com arquitetura CISC, como os Intel que usam o conjunto de instruções x86, uma instrução de soma com dois operandos, sendo um deles armazenado em um registrador (reg 1) e o outro armazenado na memória. Como em uma arquitetura do tipo CISC o endereço de memória é obtido de um complexo cálculo entre dois valores, teremos:

```
ADD     reg1, reg2 B    (para simplificar, usou-se reg1 e reg2 em vez dos nomes dos registradores da
                         arquitetura x86).
```

Isto significa que haverá um gasto de tempo para se calcular o endereço de acesso à memória, ao somar-se o valor B com o conteúdo do registrador reg2.

Em um processador RISC seria aceitável efetuar a mesma operação de soma assim:

```
MOV     reg2, B
ADD     reg1, reg2
```

No exemplo, foram usadas duas instruções em vez de apenas uma no processador CISC (com processadores RISC é normal usar mais instruções ainda), porém sua construção é mais simples e sua execução bem mais rápida, de modo que o tempo total será menor.

Assim, durante as últimas duas décadas os centros de pesquisas e universidades vêm debatendo e a indústria vem pesquisando e construindo processadores que adotam uma ou outra das arquiteturas, alguns mantendo-se com máquinas CISC e outros criando e produzindo processadores com arquitetura RISC.

11.2 CARACTERÍSTICAS DAS ARQUITETURAS CISC

Conforme já definido anteriormente, o nome CISC (Complex Instruction Set Computer) advém do fato de se considerar complexo um conjunto constituído de grande quantidade de instruções, com múltiplos modos de endereçamento, entre outras críticas.

É possível imaginar o que os projetistas desses conjuntos esperavam, em uma época inicial da computação, em que a memória era cara e pequena e, por isso, os códigos gerados pelos compiladores deveriam ser compactos e eficientes na execução.

Dessa forma, os projetistas precisavam obter boa densidade do código de máquina, ou seja, cada instrução deveria fazer muito, de modo que o programa completo tivesse poucas instruções.

O surgimento, em 1951, do conceito de microprogramação [WILK51] facilitou o trabalho de projetar instruções complexas, implementando-as em microcódigo. Além disso, apareceram nessa esteira outras vantagens, tais como: como o microcódigo reside em memória de controle, pode-se acelerar sua execução com essas memórias sendo rápidas. Além disso, como a quantidade de células da memória é usualmente potência de dois, sempre sobra espaço nas memórias para se acrescentar novos microprogramas e, assim, a criação de novas instruções é, na maioria das vezes, quase sem custo e sem aumento de espaço, facilitando a implementação do conceito de famílias de processadores.

Um bom exemplo disso é a arquitetura x86, em que foram progressivamente acrescentadas novas instruções aos processadores que iam sendo lançados (386 para 486, para Pentium, para Pentium MMX e assim por diante), sem que o projeto básico se alterasse.

Outra vantagem do emprego de microcódigo reside na rapidez da execução de instruções que estão armazenadas em uma memória (memória ROM de controle, dentro da unidade de controle) bem mais rápida que a memória convencional.

O primeiro sistema de computação lançado com microcódigo e que originou, também, o conceito de família de computadores foi introduzido pela IBM em 1964, o Sistema IBM/360. Posteriormente, a DEC (Digital Equipment Corporation) introduziu sua família de PDP, mais tarde substituída pelo sistema VAX, um dos melhores exemplos de máquina CISC. Alguns sistemas VAX chegaram a possuir mais de 300 instruções de máquina, usando mais de 15 modos de endereçamento diferentes, com instruções de 2 a 57 bytes de largura.

Pode-se concluir que os projetistas de arquiteturas CISC consideram três aspectos básicos:

- uso de microcódigo;
- construção de conjuntos com instruções completas e eficientes (completeza no conjunto);
- criação de instruções de máquina de "alto nível", ou seja, com complexidade semelhante à dos comandos de alto nível.

Colocados juntos, esses elementos de projeto nortearam a filosofia de construção dos processadores CISC por longo tempo, como a família Intel x86, os processadores AMD K e, anteriormente, os sistemas IBM e VAX.

Assim é que existem naqueles conjuntos instruções poderosas, do tipo:

CAS	- compare and swap operands	(comparar valores e trocar operandos)
RTR	- return and restore codes	(retornar e restaurar código)
SWAP	- swap register words	(trocar palavras dos registradores)

Embora os fabricantes tenham seguido suas próprias concepções, agindo de forma independente, em geral o desenvolvimento das arquiteturas CISC tende a seguir algumas regras básicas:

a) Formato de dois operandos mais comum – instruções com campos origem e destino, como a instrução

```
ADD   CX, mem    (subtrair o valor na memória do valor no registrador CX e colocar Resultado
                  no registrador CX)
```

b) Uso de modos registrador para registrador; registrador para memória e memória para registrador.

c) Uso de múltiplos modos de endereçamento para a memória, incluindo indexação para o caso de vetores.

d) Instruções com largura variável, com a quantidade de bytes variando de acordo com o modo de endereçamento utilizado.

e) As instruções requerem múltiplos ciclos de relógio para sua completa execução, além do que a quantidade desses ciclos varia de acordo com a largura das instruções. Por exemplo, se uma instrução realiza mais de um acesso à memória para buscar dois operandos, então gasta mais ciclos do que outra que só realiza um acesso.

f) O hardware possui poucos registradores devido ao fato de possuir muitas instruções com acesso à memória e por causa da limitação do espaço no chip usado para memória de controle (microcódigo), decodificação, etc.

g) Além de poucos, há também registradores especializados, como o registrador de controle (para códigos de condição ou flags, que instruções aritméticas possuem); de segmento, para o ponteiro da pilha, para tratamento de interrupção e outros.

Como é usual acontecer em qualquer área da atividade humana, é raro que algum conceito ou tecnologia importante, principalmente se interfere com muitos e, principalmente, que envolve interesses comerciais e financeiros enormes, obtenha unanimidade entre pesquisadores, técnicos, projetistas e administradores. Este é o caso da arquitetura CISC, a qual sempre foi alvo de críticas e comentários sobre desvantagens e problemas.

Neste texto não cabe posicionamento por este ou aquele fato ou tecnologia, mas sim apresentar todos os elementos possíveis das diversas tendências, no caso entre CISC e RISC.

No entanto, para se compreender o surgimento de processadores com arquitetura RISC deve-se analisar os eventuais problemas indicados para a arquitetura CISC, que levaram pesquisadores e projetistas de sistemas a criar uma alternativa, considerada por eles mais vantajosa.

Para entender melhor as raízes do surgimento da filofia RISC, pode-se mencionar alguns pontos das arquiteturas CISC citados como problemáticos por um dos criadores de máquinas RISC, David Petterson, em um de seus artigos [PATT80], induzindo ao projeto de processadores que pudessem, com sua especificação mais simples, reduzir ou eliminar os citados problemas. Na realidade, parece ter sido Patterson quem primeiro definiu as arquiteturas com muitas e poderosas instruções de CISC e sua máquina protótipo de RISC (o nome escolhido foi RISC-1):

Diferenças de velocidade entre memória e processador – no final da década de 1970, a IBM verificou que essa diferença era um problema em seus sistemas, visto que algumas operações (como aritméticas de ponto flutuante) eram realizadas por programas, acarretando muitos acessos a uma memória lenta. A solução encontrada foi criar novas instruções de máquina para executar tais operações, podendo-se acreditar que esse foi o início do aumento da quantidade de instruções nas CISC. Segundo Patterson, não parece que esta solução tenha reduzido as diferenças de velocidade.

Emprego de microcódigo – o surgimento e a real vantagem de custo/benefício do emprego de microcódigo sobre programação diretamente no hardware induziram os projetistas a criar mais e mais instruções, devido à facilidade e à flexibilidade decorrentes.

Desenvolvimento acelerado de linguagens de alto nível – na década de 1980, quando essas considerações foram elaboradas, havia um crescimento acelerado do emprego de linguagens de alto nível, o que conduzia os projetistas de processadores a incluir cada vez mais instruções de máquinas em seus produtos, com o propósito de manter um suporte adequado na compilação.

Densidade do código a ser executado – as arquiteturas CISC procuram, entre outros requisitos, obter um código compacto após a compilação, de modo a não consumir memória em excesso. Isso era possivelmente necessário em uma época em que as memórias eram caras e de reduzido tamanho. Assim, construindo conjuntos de instruções, cada uma delas mais próxima do significado do comando de alto nível, poder-se-ia obter códigos

executáveis mais densos, mais compactos, visto que a tradução de um comando de alto nível redundaria em muito poucas isntruções de máquinas. Alega Patterson, no entanto, que isto acarretaria também mais bits nas instruções (códigos de operações com mais bits devido à quantidade delas, bem como mais modos de endereçamento), o que contrabalançaria aquela pretensa vantagem (densidade do código).

Necessidade de compatibilidade com processadores anteriores – uma das metas sempre seguida pela Intel e outros fabricantes foi a de conservar a compatibilidade entre as versões de seus processadores de modo a manter o mercado cativo. Assim, o processador 486 veio com apenas algumas instruções novas e todo o código do 386 junto, de modo que códigos executáveis para o 386 rodavam também no 486, e os usuários poderiam trocar de computador sem nenhum custo adicional de compilação, etc. O mesmo aconteceu com o Pentium I, II, III e 4. Mesmo isso, embora seja um notório requisito importante de marketing, acarreta uma limitação na especificação de novas arquiteturas. Dessa forma, as arquiteturas novas só crescem em quantidade de instruções, visto que o fabricante nunca retira as instruções antigas devido ao problema de compatibilidade.

11.3 CARACTERÍSTICAS DAS ARQUITETURAS RISC

O surgimento de processadores com características bastante diferentes do que conhecemos por CISC foi resultante de trabalhos de vários pesquisadores em diferentes locais, visando encontrar possibilidades de aperfeiçoamento do desempenho dos processadores por caminhos diferentes dos adotados na arquitetura CISC. Tais pesquisadores consideravam as já citadas dificuldades e desvantagens, e portanto o escopo da nova arquitetura era projetar um processador simples e, por isso, eficaz em termos de velocidade e desempenho.

Sob este ponto de vista, mostramos de forma resumida no item 11.1 os resultados de vários estudos sobre comportamento de programas que pudessem indicar possíveis soluções para os problemas citados no item 11.2.

O desenvolvimento de arquiteturas RISC teve início por três caminhos próximos, embora conduzindo a alternativas diferentes, a saber:

1) O primeiro deles, cronologicamente, foi um projeto da IBM, desenvolvido em meados da década de 1970, pelo pesquisador daquela empresa, John Cocke, denominado IBM 801, o qual nem chegou a se tornar realidade comercial, o que acarretou pouca divulgação sobre o projeto em si, mas serviu de base para os desenvolvimentos seguintes da IBM nessa área.
2) Outra linha de pesquisa desenvolvida um pouco mais tarde (início da década de 1980) na Universidade Stanford, na Califórnia, EUA, pela equipe de John Hennessey [HENN84], redundou posteriormente nos processadores MIPS e na criação da empresa MIPS Technology Inc.
3) Finalmente, na mesma época (1980), na Universidade da Califórnia, campus de Berkeley, EUA, foram desenvolvidas pesquisas semelhantes por uma equipe liderada por David Patterson, cujos primeiros protótipos, RISC-1 e RISC-2, tornaram-se a base para o surgimento posterior dos processadores SPARC.

Mais adiante serão apresentadas algumas características descritivas dessas arquiteturas e de alguns dos processadores decorrentes.

Podem-se descrever os elementos que constituem a base da arquitetura RISC (Reduced Instruction Set Computer) através das seguintes assertivas, as quais serão comentadas a seguir:

- pequeno conjunto de instruções, todas com largura fixa;
- execução otimizada de chamada de funções;
- menor quantidade de modos de endereçamento;
- uso intenso de pipelining;
- execução rápida de cada instrução (uma por ciclo de relógio).

11.3.1 Menor Quantidade de Instruções, Todas com Largura Fixa

Talvez a característica mais marcante de um sistema RISC seja a sua tendência de possuir um conjunto de instruções menor que o das máquinas CISC de mesma capacidade. Daí o nome da arquitetura (RISC – com-

putadores com conjunto reduzido de instruções). A família SPARC, da SUN, possui cerca de 50 instruções, enquanto os VAX-11/780 tinham até 300 instruções, o Intel 80486 foi lançado com 147 instruções de máquina e os atuais Pentium possuem mais de 200 instruções.

Com menor quantidade de instruções e com cada uma delas tendo sua execução otimizada, o sistema deve produzir seus resultados com melhor desempenho, mesmo considerando que uma menor quantidade de instruções vá conduzir a programas um pouco mais longos.

Na realidade, com um conjunto reduzido de instruções de máquina obtêm-se outras vantagens, entre as quais, pode-se citar:

1) menor quantidade de transistores no chip e, conseqüentemente, menor espaço físico do VLSI e menor custo;
2) redução da complexidade do decodificador de instruções (menor largura de entrada e muito menor largura de saída), reduzindo também o tempo de decodificação;
3) menor quantidade de bits no campo código de operação da instrução, reduzindo o tamanho dos programas.

Com todas as instruções apresentando o mesmo tamanho em bits e alinhadas à largura da palavra (32 bits, por exemplo), facilita-se o trabalho de busca da instrução (*fetch cycle*), visto que ela pode ser realizada em uma única operação, e não há necessidade de verificação do seu tamanho para que o CI – Contador de Instruções possa ser corretamente incrementado, pois ele será sempre incrementado com o mesmo valor. Além disso, estando todas as instruções alinhadas por palavra e byte, não há possibilidade de uma instrução ocupar, p.ex., duas páginas de dados diferentes, o que traria problemas no acesso, por fazer o sistema operacional ter que trazer uma segunda página (tempo e espaço) por muito pouco.

Enquanto o VAX, por exemplo, tinha até 300 instruções, algumas com largura de 4 bytes, outras com 8 bytes e até instruções com 57 bytes de tamanho, os processadores SPARC possuem menos de 65 instruções, todas com 32 bits (4 bytes) de largura.

11.3.2 Execução Otimizada de Chamada de Funções

Outra característica importante da arquitetura RISC, que a distingue da arquitetura CISC, refere-se ao modo de realizar chamadas de rotinas e passagem de parâmetros. Os estudos sobre comportamento dos programas revelaram que chamadas de funções (que consomem razoável tempo do processador) requerem usualmente poucos dados, mas consomem, na transferência, demorados acessos à memória em leituras e escritas.

Enquanto em máquinas CISC a chamada de funções conduz a operações de leitura/escrita com a memória para passagem de parâmetros e recuperação de dados, nas máquinas com arquitetura RISC isto ocorre basicamente no processador, utilizando-se para isso mais registradores que as máquinas CISC; os parâmetros e variáveis são manuseados no próprio processador. A colocação de mais registradores no processador é possível devido à redução dos circuitos necessários à decodificação e à execução de instruções (porque há menor quantidade delas).

Com isso, o desempenho total do processador melhora, já que executa mais otimizadamente as chamadas de funções e estas ocorrem em quantidade apreciável na média dos programas.

11.3.3 Menor Quantidade de Modos de Endereçamento

Para facilitar o trabalho dos compiladores, o conjunto de instruções de máquinas CISC tende a possuir muitos modos de endereçamento (embora os atuais Pentium possuam menos modos, os processadores da família VAX-11 tinham até 22 modos) de endereçamento.

Uma simples instrução de soma pode ser realizada com os operandos localizados de diversos modos: podem-se somar valores que estão armazenados em registradores; outra instrução pode realizar a mesma soma, porém com um operando na memória e outro em um registrador, ou ainda uma outra instrução pode realizar a operação de soma com os dois operandos armazenados na memória.

No entanto, já foi mencionado que os estudos realizados sobre comportamento de compiladores e sobre a geração de código executável concluíram que os compiladores não usam a maioria dos modos de endereçamento disponíveis, nem mesmo a maioria das instruções existentes para "simplificar" seu trabalho.

Por isso, nas máquinas RISC a busca por soluções mais simples conduziu à criação, de um modo geral, de apenas dois tipos de instruções para acesso à memória: LOAD/STORE (LOAD – transfere o dado da memória para um dos registradores do processador; STORE – operação inversa, que transfere um dado de um dos registradores para a memória). Essas instruções utilizam somente o modo direto (ver modos de endereçamento no Cap. 8) e demais operações no processador (as operações matemáticas). Esta técnica simplifica consideravelmente o projeto e a implementação das instruções, reduzindo ainda mais os ciclos de relógio necessários à sua realização.

11.3.4 Modo de Execução com *Pipelining*

Uma das características mais relevantes da arquitetura RISC é o uso altamente produtivo de pipelining, obtido em face do formato simples e único das instruções de máquina.

Conforme poderemos observar no Apêndice D, a técnica pipelining funciona mais efetivamente quando as instruções são todas bastante semelhantes, pelo menos no que se refere ao seu formato e complexidade, o que redunda em mesmo tempo de excução de suas diversas etapas.

Isto é verdade se imaginarmos que os estágios de uma linha de montagem devem consistir em tarefas semelhantes em tempo e forma para que a produtividade seja maior. Porque não é interessante se um estágio terminar antes do outro e tiver que esperar a conclusão do estágio mais demorado, pois nesse caso perde-se a vantagem da linha de montagem (que pára em espera). O objetivo é cada instrução completar um estágio pipeline em um ciclo do relógio, embora isso nem sempre seja alcançado.

Por exemplo, uma linha de montagem de fabricação de um determinado objeto consome uma hora para terminar a fabricação de um único objeto e está dividida em quatro estágios. Se cada estágio estiver organizado de modo que suas tarefas para o processo de fabricação de um objeto durem aproximadamente 15 minutos, então o sistema será altamente produtivo, completando-se um objeto a cada 15 minutos.

No entanto, se as tarefas dos estágios estiverem desbalanceadas e, por exemplo, o estágio 1 durar 20 minutos, o estágio 2 durar 30 minutos e os restantes estágios completarem em 5 minutos cada um, então a produtividade desejada para a linha de montagem estará comprometida. Nesse caso, um segundo objeto será iniciado 20 minutos após o primeiro ter iniciado (e não 15 minutos); como o segundo estágio completa-se em 30 minutos, então o segundo objeto somente poderá iniciar o segundo estágio no minuto 50 e não no 40, como deveria, ficando 10 minutos ocioso. Enquanto o primeiro objeto está completo em 60 minutos, o segundo somente se completará no minuto 90, 30 minutos após, e não em 15 minutos, como aconteceria se os estágios tivessem duração igual.

Se continuássemos o processo (e o leitor é instado a fazê-lo, criando uma linha de tempo para, digamos, quatro objetos e seguindo os tempos estabelecidos para os estágios), iríamos constatar que a situação iria piorar à medida que os objetos fossem iniciados, deixando-se muito tempo de ociosidade entre os estágios devido à desigualdade entre eles.

Conforme já observado, projetar processadores que executam várias instruções quase que totalmente em paralelo é uma técnica bastante eficaz para acelerar o desempenho dos processadores, reduzindo o tempo de execução das instruções para poucos ciclos.

11.3.5 Execução de Cada Instrução em um Ciclo de Relógio

Se o uso correto e eficaz de pipelining (tecnologia de linha de montagem) é considerado uma das características mais importantes do projeto de uma arquitetura RISC, a definição de execução de uma instrução por ciclo de relógio é ainda mais importante, sendo um dos requisitos essenciais desse tipo de arquitetura, segundo aqueles que primeiro estabeleceram suas bases.

Na realidade, um dos aspectos mais negativos apontados para os processadores CISC reside no longo tempo de execução de certas instruções (quase todas) devido, entre outras coisas, ao uso de microcódigo, o que implica a interpretação de cada microoperação, com o decorrente atraso na execução total da instrução.

Característica	Considerações
Menor quantidade de instruções que as máquinas CISC	• Simplifica o processamento de cada instrução e torna este item mais eficaz. • Embora o processador RS/600 possua 184 instruções, ainda assim é bem menos que as 303 instruções dos sistemas VAX-11. Além disso, a maioria das instruções é realizada em 1 ciclo de relógio, o que é considerado o objetivo maior dessa arquitetura.
Execução otimizada de chamada de funções	• As máquinas RISC utilizam os registradores da UCP (em maior quantidade que os processadores CISC) para armazenar parâmetros e variáveis em chamadas de rotinas e funções. Os processadores CISC usam mais a memória para a tarefa.
Menor quantidade de modos de endereçamento	• As instruções de processadores RISC são basicamente do tipo Load/Store, desvio e de operações aritméticas e lógicas, reduzindo com isso seu tamanho. • A grande quantidade de modos de endereçamento das instruções de processadores CISC aumenta o tempo de execução das mesmas.
Utilização em larga escala de *pipelining*	• Um dos fatores principais que permite aos processadores RISC atingir seu objetivo de completar a execução de uma instrução pelo menos a cada ciclo de relógio é o emprego de *pipelining* em larga escala.

Figura 11.4 Características de processadores RISC.

Processadores RISC, por outro lado, por serem constituídos de poucas instruções e todas elas (com exceção, talvez, das instruções LOAD e STORE, que acessam a memória) simples, não requerem microcódigos.

A tabela da Fig. 11.4 apresenta um resumo das características básicas das arquiteturas RISC que as distinguem das CISC e contribuem para o melhor desempenho do hardware em termos gerais.

11.4 RISC × CISC

A tabela da Fig. 11.5 mostra exemplos classicos de características de máquinas CISC (IBM, Intel, VAX), comparando-as com arquiteturas RISC.

Embora haja atualmente um número razoável de adeptos das máquinas que possuem arquitetura RISC, também há, e em grande quantidade, aqueles que relacionam diversas desvantagens desses processadores, advogando em favor da arquitetura CISC.

Vários podem ser os temas para discussão sobre RISC e CISC, um dos quais se refere ao desempenho do processador na execução de um programa. De modo geral, os vendedores e outros pesquisadores tendem a

Características	RISC		CISC	
	MIPS R4000	RS/6000	VAX11/780	INTEL 486
Quantidade de instruções	94	183	303	235
Modos de endereçamento	1	4	22	11
Largura de cada instrução (bytes)	4	4	2-57	1-12
Quantidade de registradores de emprego geral	32	32	16	8

Figura 11.5 Características de alguns processadores RISC e CISC.

medir o desempenho através de programas de teste (*benchmarks*), já discutidos no item anterior. No entanto, verificamos que os referidos programas possuem uma série de complicações na interpretação de seus resultados em função do tipo de ambiente que utilizaram e da natureza dos testes.

Em princípio, os defensores da arquitetura CISC propugnam que instruções mais complexas redundarão em um código-objeto menor (as instruções de máquina, sendo mais complexas, se aproximam em definição dos comandos da linguagem de alto nível que está sendo compilada), o que reduz o consumo de memória (menos instruções), com reflexos no custo do sistema.

Isso não é necessariamente correto se considerarmos que uma menor quantidade de instruções nem sempre acarreta menor quantidade de bits (e é a quantidade efetiva de bits que consome menos memória e a menor custo). Se cada instrução CISC possuir mais operandos que as instruções RISC e se cada um de seus operandos ocupar uma boa quantidade de bits na instrução, então poderemos ter um programa CISC maior em bits do que um programa em máquina RISC, apesar de o programa para o processador RISC possuir maior quantidade de instruções.

Por exemplo, um programa escrito para rodar em um processador CISC pode gastar 150 instruções de máquina; cada uma das instruções possui código de operação de 8 bits, podendo ser de um, de dois e três operandos. Cada campo operando ocupa 18 bits e ainda há um campo para outras ações, com 4 bits de tamanho. Em média, as instruções têm um total de 50 bits. Um programa para realizar o mesmo problema, escrito para rodar em um processador RISC, pode ter 220 instruções, que em média ocupam 32 bits.

As instruções são, em sua esmagadora maioria, de dois operandos, porém os operandos são valores em registradores e, por isso, as instruções não consomem muitos bits para endereçar os dois registradores. Como há relativamente poucas instruções, elas têm um campo código de operação de 6 bits.

O programa para a máquina CISC gastaria 7.500 bits, enquanto o programa para a máquina RISC, mesmo possuindo mais 70 instruções que o do processador CISC, consumiria 7.040 bits. Trata-se, evidentemente, de um exemplo simples porém elucidativo, porque os valores apresentados estão próximos da realidade das máquinas atuais.

Outro ponto de debate se refere à rapidez da execução de um programa. Os defensores da arquitetura CISC alegam que estas máquinas executam mais rapidamente os programas escritos em linguagem de alto nível devido à pouca quantidade de códigos binários executáveis. No entanto, o tempo que cada instrução leva para ser executada nem sempre conduz à confirmação dessa assertiva.

Máquinas RISC tendem a executar instruções bem mais rápido porque:

a) as instruções possuem C.Op. com menor quantidade de bits (pois o conjunto de instruções é menor) e, portanto, o tempo de decodificação é menor que o das máquinas CISQ; e

b) as instruções são executadas diretamente pelo hardware e não por um microprograma. Conquanto um processador microprogramado traga mais flexibilidade ao projeto das máquinas, ele também acarreta uma sobrecarga adicional de interpretação de cada instrução. Máquinas RISC não são microprogramadas e, assim, tendem a executar as instruções de modo mais rápido.

Processadores RISC são também otimizados para operações de uma única tarefa devido ao grande número de registradores que possuem e à grande quantidade de estágios de *pipelining*. Nesses casos, a melhor maneira de obter um bom desempenho dos processadores RISC é executar um programa de teste (um *benchmark*), o qual possui exatamente esta característica: um grande número de operações similares, em uma única tarefa. Interessados em processamento científico podem se apoiar mais nesses programas de teste porque o processamento que fazem é similar ao dos programas usuais de teste. Mas o mesmo não se pode dizer de programas comerciais, que utilizam muita E/S (tarefa que, por exemplo, os programas de teste não realizam).

Neste ponto, e antes de serem apresentados alguns exemplos de arquiteturas clássicas RISC, deve-se observar que a discussão e detalhamento de características de processadores que seguem a filosofia CISC e os que seguem a filosofia RISC é atualmente bem menos crítica do que foi em anos anteriores, quando havia realmente uma nítida distinção entre ambas.

Com o passar do tempo, o avanço da tecnologia em hardware, principalmente, modificou a visão de alguns projetistas e as adaptações foram surgindo de ambas as partes, de modo que atualmente não se pode afirmar

com absoluta certeza que um determinado processador segue rigorosamente a linha RISC nem que outro segue rigorosamente a linha CISC (talvez até mais um pouco neste último caso).

Assim é que, os últimos processadores Intel possuem um núcleo de execução RISC, assim como os processadores de 64 bits, Itanium, seguem, em grande parte, as características definidas para um componente RISC. E, é bem verdade, que processadores Power sempre possuíram uma quantidade apreciável de instruções (fugindo ao padrão original RISC), embora todas com largura fixa.

Além disso, o uso de técnicas de superpipeline (processadores MIPS) e superescalar (Sparc, PowerPC, etc), embora acarretem desempenho muito bom, não contribuem para simplicidade do processador, como se preconizava para a filosofia RISC.

11.5 EXEMPLOS DE ARQUITETURAS RISC

Conforme já foi mencionado anteriormente, o desenvolvimento de processadores com definição de arquitetura do tipo RISC aconteceu, pelo menos inicialmente, em três vertentes, uma evoluindo no âmbito das áreas de pesquisa e produção da IBM e outras duas oriundas de pesquisa em universidades americanas, financiadas pelo governo e pela iniciativa privada, uma em Berkeley e outra em Stanford, as quais geraram linhas de processadores e empresas correspondentes, respectivamente, o SPARC (Sun Microsystems) e o MIPS (MIPS Tecnology). Posteriormente, a Intel e a HP também vieram a produzir processadores com arquitetura RISC, e atualmente o próprio Pentium 4 tem um núcleo interno de execução nos moldes RISC, bem como a arquitetura IA-64 segue o padrão de simplicidade e paralelismo RISC. A seguir, vamos apresentar algumas considerações sobre esses desenvolvimentos, com exemplos sobre alguns dos produtos de cada um.

A tabela da Fig. 11.6 apresenta uma distribuição percentual do mercado americano de estações de trabalho RISC em 1993, segundo pesquisa realizada pela empresa Computer Intelligence InfoCorp.

Fabricante/Vendedor	Percentagem de máquinas vendidas/alugadas/leased
Sun Microsystems	44,4%
Hewlett-Packard	21,3%
IBM	12,6%
DEC	6,6%
Silicon Graphics	6,5%
Outros	8,6%

Figura 11.6 Distribuição das estações de trabalho no mercado americano em 1993. (Fonte: Computer Intelligence InfoCorp.)

11.5.1 O Desenvolvimento da Arquitetura RISC na IBM

É muitas vezes difícil estabelecer na indústria ou mesmo em outras atividades o que foi realmente feito primeiro, o que foi descoberto ou criado ou inventado em primeiro lugar. Até bem pouco tempo tinha-se certeza de que o telefone tinha sido inventado por Alexander G. Bell, e agora sabe-se que 10 anos antes o cientista Antonio Meucci tinha apresentado seu sistema, tendo o Congresso norte-americano reconhecido essa primazia em 2002.

Da mesma forma, sobre o primeiro sistema ou o primeiro desenvolvimento do tipo RISC a literatura tem mencionado a IBM, em 1970, e algum tempo depois as pesquisas de Berkeley e Stanford (1980).