Organização de Computadores

Prof. Robson de Souza

Nível de Arquitetura do Conjunto de Instrução (ISA)

O nível de arquitetura do conjunto de instrução (ISA – Instruction Set Architecture) está posicionado entre o da microarquitetura e o do sistema operacional. Historicamente, ele foi desenvolvido antes de quaisquer outros níveis e, na verdade, na origem era o único. Até hoje, não é raro ouvir esse nível ser chamado de "a arquitetura" de uma máquina ou, às vezes (incorretamente), como a "linguagem de montagem".
O nível ISA tem um significado especial que o torna importante para arquitetos de sistemas: é a interface entre o software e o hardware. Embora seja possível o hardware executar diretamente programas escritos em C, C++, Java ou alguma outra linguagem de alto nível, isso não seria uma boa ideia, até porque a maioria dos computadores deve ser capaz de executar programas escritos em várias linguagens, e não apenas em uma.

A abordagem de base adotada por todos os projetistas de sistemas é traduzir programas escritos em várias linguagens de alto nível para uma forma intermediária comum – nível ISA – e construir hardware que possa executar programas diretamente no nível ISA. O nível iSA define a interface entre os compiladores e o hardware. É a linguagem que ambos têm de entender.

Princípios iniciais

Primeiramente, é preciso entender o que é necessário para desenvolver uma ISA de qualidade, ou seja, tentar responder à seguinte pergunta: O que faz uma ISA ser boa? Como resposta, é possível afirmar que há dois fatores primordiais.

Primeiro, ela deve definir um **conjunto de instruções que pode ser executado com eficiência em tecnologias atuais e futuras**, resultando em projetos efetivos em custo por várias gerações. Um mau projeto é mais difícil de realizar e pode exigir um número muito maior de portas para implementar um processador e mais memória para executar programas.

Segundo, uma boa ISA deve **fornecer um alvo claro para o código compilado**. Regularidade e completude de uma faixa de opções são aspectos importantes que nem sempre estão presentes em uma ISA. Em resumo, uma vez que a ISA é a interface entre hardware e software, ela tem de contentar os projetistas de hardware (fácil de implementar com eficiência) e satisfazer os projetistas de software (fácil de gerar bom código para ela).

Propriedades do nível ISA

Em princípio, o nível ISA é definido pelo modo como a máquina se apresenta a um programador de linguagem de máquina. Em resumo, podemos dizer que o código de nível ISA é o que um compilador de linguagem de mais alto nível produz.

Para produzir código de nível ISA, o escritor de compilador tem de saber qual é o modelo de memória, quais e quantos são os registradores, quais tipos de dados e instruções estão disponíveis, e assim por diante. **O conjunto de todas essas informações define o nível ISA**.

Todos os computadores dividem a memória em células que têm endereços consecutivos. O tamanho de célula mais comum no momento é 8 bits, mas células de 1 bit a 60 bits já foram usadas no passado. Uma célula de 8 bits é denominada byte (ou octeto). A razão para usar bytes de 8 bits é que os caracteres ASCII têm 7 bits, de modo que um caractere ASCII (mais um bit de paridade) se encaixa em um byte.

Em geral, os bytes são agrupados em palavras de 4 bytes (32 bits) ou 8 bytes (64 bits) com instruções disponíveis para manipular palavras inteiras. Muitas arquiteturas precisam que as palavras sejam alinhadas em seus limites naturais; assim, por exemplo, uma palavra de 4 bytes pode começar no endereço 0, 4, 8 etc., mas não no endereço 1 ou 2. De modo semelhante, uma palavra de 8 bytes pode começar no endereço 0, 8 ou

16, mas não no endereço 4 ou 6.

O alinhamento costuma ser exigido porque memórias funcionam com mais eficiência desse modo. O Core i7, por exemplo, que busca 8 bytes por vez na memória, usa uma interface DDR3, que admite apenas acessos alinhados em 64 bits. Assim, o Core i7 não poderia fazer uma referência desalinhada à memória nem que quisesse, porque a interface de memória exige endereços que sejam múltiplos de 8.

Outro aspecto do modelo de memória de nível ISA é a semântica da memória. É muito natural esperar que uma instrução LOAD que ocorre após uma instrução STORE, e que referencia o mesmo endereço, retornará o valor que acabou de ser armazenado. Todavia, em muitos projetos, as microinstruções são reordenadas. Assim, há um perigo real de que a memória não terá o comportamento esperado. O problema fica ainda pior em um multiprocessador, no qual cada uma das várias CPUs envia uma sequência de requisições de escrita e leitura (talvez reordenadas) a uma memória compartilhada.

Projetistas de sistemas podem adotar qualquer uma de diversas técnicas para resolver esse problema. Em um extremo, todas as requisições de memória podem ser serializadas, portanto, cada uma é concluída antes de a próxima ser emitida. Essa estratégia prejudica o desempenho, mas resulta na semântica de memória mais simples (todas as operações são executadas estritamente na ordem do programa).

No outro extremo, não são dadas garantias de espécie alguma. Para forçar uma ordenação na memória, o programa deve executar uma instrução SYNC, que bloqueia a emissão de todas as novas operações de memória até que todas as anteriores tenham sido concluídas. Esse esquema atribui uma grande carga aos compiladores, porque eles têm de entender, com detalhes, como a microarquitetura subjacente funciona, embora dê aos projetistas de hardware a máxima liberdade para otimizar a utilização da memória.

Também são possíveis modelos de memória intermediários, nos quais o hardware bloqueia automaticamente a emissão de certas referências à memória, mas não bloqueia outras.

Tipos de dados

Todos os computadores precisam de dados. Os dados têm de ser representados de alguma forma específica no interior do computador. No nível iSA, são usados vários tipos de dados diferentes.

*Tipos de dados numéricos

Os tipos de dados podem ser divididos em duas categorias: numéricos e não numéricos. O principal entre os tipos de dados numéricos são os inteiros. Eles podem ter muitos comprimentos, em geral 8, 16, 32 e 64 bits. inteiros contam coisas, identificam coisas (por exemplo, números de contas correntes) e muito mais. A maioria dos computadores modernos armazena inteiros em notação binária de complemento de dois, embora outros sistemas já tenham sido usados no passado.

Alguns computadores suportam inteiros **sem sinal**, bem como inteiros **com sinal**. No caso de um inteiro sem sinal, não há bit de sinal e todos os bits contêm dados. Esse tipo de dado tem a vantagem de um bit extra, portanto, por exemplo, uma palavra de 32 bits pode conter um único inteiro sem sinal na faixa de 0 a $2^{32} – 1$, inclusive. Ao contrário, um inteiro de 32 bits com sinal, representado por complemento de dois, só pode manipular números até $2^{31} – 1$, mas, é claro, também pode manipular números negativos.

Para números que não podem ser expressos como um inteiro, como 3.5, são usados números de ponto flutuante. Eles têm comprimentos de 32, 64 ou, às vezes, 128 bits. A maioria dos computadores tem instruções para efetuar aritmética de ponto flutuante. Muitos deles têm registradores separados para conter operandos inteiros e para conter operandos de ponto flutuante.

*Tipos de dados não numéricos

Computadores modernos são usados com frequência para aplicações não numéricas, como e-mail, navegar pela Web, fotografia digital e criação e reprodução de multimídia. Para essas aplicações, são necessários outros tipos de dados, que muitas vezes são suportados por instruções de nível ISA. Nesse caso, é clara a

importância dos caracteres, embora nem todos os computadores ofereçam suporte de hardware para eles. Os códigos mais comuns são ASCII e Unicode. Eles suportam caracteres de 7 bits e de 16 bits, respectivamente.

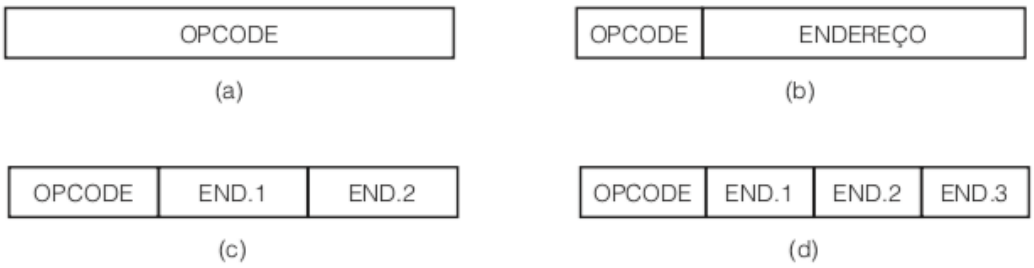
Não é incomum que o nível ISA tenha instruções especiais destinadas a manipular **cadeias de caracteres**, isto é, carreiras consecutivas de caracteres. Essas cadeias às vezes são delimitadas por um caractere especial na extremidade. Como alternativa, um campo de comprimento de cadeia pode ser usado para monitorar essa extremidade. As instruções podem executar cópia, busca, edição e outras funções nas cadeias.

Valores booleanos também são importantes. Um valor booleano pode assumir um de dois valores: verdadeiro ou falso. Em teoria, um único bit pode representar um booleano, com 0 para falso e 1 para verdadeiro (ou vice-versa). Na prática, é usado um byte ou uma palavra por valor booleano, porque bits individuais em um byte não têm seus endereços próprios e, portanto, são difíceis de acessar. Um sistema comum usa a seguinte convenção: 0 significa falso e qualquer outra coisa significa verdadeiro.

Formatos de instrução

Uma instrução consiste em um opcode, normalmente em conjunto com alguma informação adicional, tais como de onde vêm os operandos e para onde vão os resultados. O tópico geral que trata de especificar onde os operandos estão (isto é, seus endereços) é denominado **endereçamento**. A figura abaixo mostra alguns diferentes formatos de instruções.

Quatro formatos comuns de instrução: (a) Instrução sem endereço. (b) Instrução de um endereço. (c) Instrução de dois endereços. (d) Instrução de três endereços.



Fonte: (Tanenbaum e Austin, 2013)

Em algumas máquinas, todas as instruções têm o mesmo comprimento; em outras, pode haver muitos comprimentos diferentes. instruções podem ser mais curtas, mais longas ou do mesmo comprimento da palavra. Ter instruções do mesmo comprimento da palavra é mais simples e facilita a decodificação, mas muitas vezes desperdiça espaço, já que, desse modo, todas as instruções precisam ser tão longas quanto a mais longa.

Modos de endereçamento

*Endereçamento imediato

O modo mais simples de uma instrução especificar um operando é a parte da instrução referente ao endereço conter o operando de fato em vez de um endereço ou outra informação que descreva onde ele está. Tal operando é denominado **operando imediato** porque ele é automaticamente buscado na memória, ao mesmo tempo em que a própria instrução. A figura abaixo mostra uma instrução imediata para carregar o registrador R1 com a constante 4

Fonte: (Tanenbaum e Austin, 2013)

O endereçamento imediato tem a vantagem de não exigir uma referência extra à memória para buscar o operando. A desvantagem é que só uma constante pode ser fornecida desse modo. Além disso, o número de valores é limitado pelo tamanho do campo.

*Endereçamento direto

Um método para especificar um operando na memória é dar seu endereço completo. Esse modo é denominado **endereçamento direto**. Assim como o imediato, o endereçamento direto tem uso restrito: a instrução sempre acessará exatamente a mesma localização de memória. Portanto, embora o valor possa mudar, sua localização não pode. Assim, o endereçamento direto só pode ser usado para acessar variáveis globais cujos endereços sejam conhecidos no momento da compilação.

*Endereçamento de registrador

Endereçamento de registrador é conceitualmente o mesmo que endereçamento direto, mas **especifica um registrador** em vez de uma localização de memória. Como os registradores são tão importantes (pelo acesso rápido e endereços curtos), esse modo de endereçamento é o mais comum na maioria dos computadores. Muitos compiladores fazem todo o possível para determinar quais variáveis serão acessadas com maior frequência (por exemplo, o índice de um laço) e as colocam em registradores. Esse modo de endereçamento é conhecido simplesmente como **modo registrador**.

*Endereçamento indireto de registrador

Nesse modo, o operando que está sendo especificado vem da memória ou vai para ela, mas seu endereço não está ligado à instrução, como no endereçamento direto. Em vez disso, está contido em um registrador. Quando um endereço é usado dessa maneira, ele é denominado **ponteiro**.
Uma grande vantagem do endereçamento indireto de registrador é que ele pode referenciar a memória sem pagar o preço de ter um endereço de memória completo na instrução. Além disso, também pode usar diferentes palavras de memória em diferentes execuções da instrução.

*Endereçamento indexado

Muitas vezes, é útil poder referenciar palavras de memória cujo deslocamento em relação a um registrador é conhecido. **Endereçamento indexado** é o nome que se dá ao endereçamento de memória que fornece um registrador (explícito ou implícito) mais um deslocamento constante.

*Endereçamento de base indexado

Algumas máquinas têm um modo de endereçamento no qual o endereço de memória é calculado somando dois registradores mais um deslocamento (opcional). Esse modo às vezes é denominado **endereçamento de base indexado**. Um dos registradores é a base e o outro é o índice. Por exemplo:

```
MOV R4, (R2+R5)
```

*Endereçamento de pilha

É muito desejável que as instruções de máquina sejam as mais curtas possíveis. O limite final na redução de comprimentos de endereços é não ter endereços. Como já foi estudado, instruções de zero endereço, como IADD, são possíveis em conjunção com uma pilha.

É uma antiga tradição da matemática colocar o operador entre os operandos, como em x + y, em vez de após os operandos, como em x y +. A forma com o operador entre os operandos é denominada **infixa**. A forma com o operador após os operandos é denominada **pós-fixa ou notação polonesa invertida**, que deve seu nome ao lógico polonês J. Lukasiewicz (1958), pesquisador das propriedades dessa notação.

A notação polonesa invertida tem diversas vantagens sobre a notação infixa para expressar fórmulas algébricas. **Primeiro,** qualquer fórmula pode ser expressa sem parênteses. **Segundo**, ela é conveniente para

avaliar fórmulas em computadores com pilhas. **Terceiro**, operadores infixos têm precedência, o que é arbitrário e indesejável.
Para ficar mais claro, observe a figura abaixo que mostra alguns exemplos de expressões infixas e seus equivalentes na notação polonesa invertida.

Notação infixa	Notação polonesa invertida
A + B × C	A B C × +
A × B + C	A B × C +
A × B + C × D	A B × C D × +
(A + B) / (C − D)	A B + C D − /
A × B / C	A B × C /
((A + B) × C + D) / (E + F + G)	A B + C × D + E F + G + /

Fonte: (Tanenbaum e Austin, 2013)

A notação polonesa invertida é a ideal para avaliar fórmulas em um computador com uma pilha. A fórmula consiste em n símbolos, cada um sendo um operando ou um operador. O algoritmo para avaliar uma fórmula em notação polonesa invertida é simples. Examine a cadeia da notação da esquerda para a direita. Quando encontrar um operando, passe-o para a pilha. Quando encontrar um operador, execute a instrução correspondente.

Referências bibliográficas:

TANENBAUM, Andrew S. Organização Estruturada de Computadores, 2007, 5ª Edição.

TANENBAUM, Andrew S. AUSTIN, Todd; Organização Estruturada de Computadores, 2013, 6ª Edição.