UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA -
ESCOLA SUPERIOR DE TECNOLOGIA - EST

JACKSON KELVIN DE SOUZA

JOSE EDUARDO MACHADO VAZ

ARM VERSUS x86

MANAUS-AM

2015

JACKSON KELVIN DE SOUZA

JACKSON KELVIN DE SOUZA

JOSE EDUARDO MACHADO VAZ

PROFESSOR: CARLOS MAURICIO S. FIGUEIREDO

MATERIA: ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

MANAUS - AM

2015

Introdução

RISC é uma linha de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções que levam aproximadamente a mesma quantidade de tempo para serem executadas. Muitos dos microprocessadores modernos são RISCs, por exemplo DEC Alpha, SPARC, MIPS, e PowerPC. Os computadores atuais misturam as duas arquiteturas, criando o conceito de arquitetura híbrida, incorporando os conceitos das duas arquiteturas e a inclusão de um núcleo RISC aos seus processadores. O tipo de microprocessador mais comum em desktops, o x86, é mais semelhante ao CISC do que ao RISC, embora chips mais novos traduzam instruções x86 baseadas em arquitetura CISC em formas baseadas em arquitetura RISC mais simples, utilizando prioridade de execução.

Os processadores baseados na computação de conjunto de instruções reduzidas não têm micro-programação, as instruções são executadas diretamente pelo hardware. Como característica, esta arquitetura, além de não ter micro código, tem o conjunto de instruções reduzidas, bem como baixo nível de complexidade.

A idéia foi inspirada pela descoberta de que muitas das características incluídas na arquitetura tradicional de processadores para ganho de desempenho foram ignoradas pelos programas que foram executados neles. Mas o desempenho do processador em relação à memória que ele acessava era crescente. Isto resultou num número de técnicas para otimização do processo dentro do processador, enquanto ao mesmo tempo tentando reduzir o número total de acessos à memória.

CISC é uma linha de arquitetura de processadores capaz de executar centenas de instruções complexas diferentes sendo, assim, extremamente versátil. Exemplos de processadores CISC são os 386 e os 486 da Intel.

Os processadores baseados na computação de conjunto de instruções complexas contêm uma micro-programação, ou seja, um conjunto de códigos de instruções que são gravados no processador, permitindo-lhe receber as instruções dos programas e executá-las, utilizando as instruções contidas na sua micro-programação. Seria como quebrar estas instruções, já em baixo nível, em diversas instruções mais próximas do hardware (as instruções contidas no microcódigo do processador). Como característica marcante esta arquitetura contém um conjunto grande de instruções, a maioria deles em um elevado grau de complexidade.

Examinando do ponto de vista um pouco mais prático, a vantagem da arquitetura CISC é que já temos muitas das instruções guardadas no próprio processador, o que facilita o trabalho dos programadores de linguagem de máquina; disponibilizando, assim, praticamente todas as instruções que serão usadas em seus programas. Os processadores CISC têm a vantagem de reduzir o tamanho do código executável por já possuirem muito do código comum em vários programas, em forma de uma única instrução.

Porém, do ponto de vista da performance, os CISCs têm algumas desvantagens em relação aos RISCs, entre elas a impossibilidade de se alterar alguma instrução *composta* para se melhorar a performance. O código equivalente às instruções *compostas* do CISC pode ser escrito nos RISCs da forma desejada, usando um conjunto de instruções

simples, da maneira que mais se adequar. Sendo assim, existe uma disputa entre tamanho do código X desempenho.

Arquitetura ARM

A Arquitetura ARM (primeiramente Acorn RISC Machine, posteriormente Advanced RISC Machine) é uma arquitetura de processador de 32 bits usada principalmente em sistemas embarcados. São processadores que visam a simplificação das instruções, com o intuito de atingir a máxima eficiência por ciclo, podendo realizar tarefas menores com ciclos mais curtos, e uma maior ordenação das operações dentro do núcleo de processamento. Muito usada na indústria e na informática, seu desenvolvimento se deu visando obter o melhor desempenho possível, com a limitação de ser simples, ocupar pouca área e ter baixo consumo de energia.

Os processadores ARM são conhecidos pela sua versatilidade, pois possuem poucas instruções para programação. São encontrados em PDAs, telefones celulares, calculadoras, periféricos de computador, equipamentos POS e aplicações industriais.

O padrão RISC do processador permite que estes processadores tenham menos transístores que processadores CISC (x86). Essa abordagem reduz custos, liberação de calor e consumo de energia. Essas são características desejáveis para dispositivos portáteis, como smartphones, laptops, tablets e outros dispositivos embarcados. Uma estrutura mais simples facilita a criação de multi-core CPUs, o que impacta na redução de custos de produção. Os processadores ARM são 90% dos processadores embarcados RISC de 32 bits.

O processador ARM possui sete modos de operação que podem ser intercambiados através do software, interrupções externas e processamento de execuções. Normalmente as aplicações são executadas a nível de usuário. Enquanto o processador esta no modo usuário o programa sendo executado é incapaz de acessar alguns recursos protegidos do sistema ou mudar de modo.

Os outros modos além do modo usuário são denominados modos privilegiados. Eles tem acesso completo aos recursos do sistema e podem mudar de modo livremente.

Registradores: Os registradores podem ser utilizados para manipular dados de um byte, de meia palavra (16 bits) ou de uma palavra completa (32 bits). Quando instruções de um byte são utilizadas somente o byte menos significativo é utilizado. Quando instruções de meia palavra são utilizadas somente a palavra menos significativa é utilizada. Ao fazer referencia a algum destes registros de propósito específico deve-se sempre levar em consideração qual o modo de operação corrente. Um outro registrador importante é o CPSR (Current Processor Status Register), que carrega informações sobre o estado corrente do processador, inclusive o modo de execução atual.

Características da arquitetura ARM:

- Arquitetura Load-Store: as instruções somente processarão (soma, subtração, etc) valores que estiverem nos registradores e sempre armazenarão os resultados em algum registrador.

- Instruções fixas de 32 bits de largura (com exceção das instruções Thumb compactas de 16 bits) alinhadas em 4 bytes consecutivos da memória, com

execução condicional, com poderosas instruções de carga e armazenamento de múltiplos registradores, capacidade de executar operações de deslocamento e na ULA com uma única instrução executada em um ciclo de clock .

- Formato de instruções de 3 endereços (isto é, os dois registradores operandos e o registrador de resultado são independentemente especificados)
- 15 registradores de 32 bits para uso geral
- Manipulação de periféricos de I/O como dispositivos mapeados na memória com suporte à interrupções.
- Conjunto de instruções aberto a extensões através de co-processador, incluindo a adição de novos registradores e tipos de dados ao mInodelo do programador.
- Pipelines de 3 a 15 estágios.
- Baixo Consumo de energia;
- Tamanho do núcleo reduzido;

Interrupções são definidas por configurações programáveis. FIQ: Fast Interrupt Request, maior prioridade. IRQ: Vectorred Interupt Request, intermediária (0 à 15). Não - Vectored Interupt Request, menor prioridade. As prioridades das interrupções dos diversos dispositivos são ajustadas dinamicamente

Bloco de conexão de pinos, este bloco permite selecionar pinos do microcontrolador que possuem mais que uma função. registros de configuração controlam os multiplexadores para permitir a conexão entre os pinos e os periféricos no chip. Periféricos devem ser conectados a pinos apropriados antes de serem habilitados e antes de qualquer interrupção relacionada seja ativada. Ativação de qualquer função periférica que não é mapeada para um pino relacionado devem ser consideradas indefinidas.

Propósitos gerais Paralell I/O e Fast I/O: pinos que não são conectados a específicas funções periféricas são controlados pelos registros do GPIO. Pinos podem ser configurados dinamicamente como entrada ou saída. Separar os registros permite configurar ou limpar qualquer número de saídas simultaneamente, o valor do registro de saída pode ser lido novamente, bem como os estados atuais das portas. No paralell e Fast I/O há controle de direção individual dos bits e todas as I/O viram input no reset.

A arquitetura x86

A linguagem de montagem x86 é a família de linguagens de montagem de compatibilidade reversa para a classe de processadores x86, que inclui a série Pentium da Intel e a série Athlon da AMD. Como todas linguagens de montagem, ela utiliza mnemônicas curtas para representar as operações fundamentais que a CPU em um computador pode realizar. Compiladores muitas vezes produzem código de montagem como um passo intermediário ao traduzir um programa de alto nível para código de máquina. Lembrada como uma linguagem de programação, a programação de

montagem é especificada à máquina e de baixo nível, com certeza o mais baixo nível possível, adereçando diretamente hardware, também conhecido como metal descoberto. Linguagens de montagens são, portanto, principalmente usadas para aplicações detalhas ou de tempo crítico como inicializadores, núcleos de sistemas operativos, e controladores de dispositivos, assim como para sistemas operativos de tempo real ou pequenos sistemas embarcados, embora seja perfeitamente possível escrever um software aplicativo em um dialeto apropriado (determinado pela escolha do montador) da linguagem de montagem com um montador que compila para a arquitetura alvo.

Mnemônicas e opcodes

Cada instrução de montagem x86 é representada por uma mnemônica, que geralmente com um ou mais operadores, traduz para um ou mais bytes, chamados opcode; a instrução NOP traduz para 0x90, por exemplo e a instrução HLT traduz para 0xF4. Há opcodes sem nenhuma mnemônica documentada, que diferentes processadores podem interpretar de maneiras diferentes, um programa usando elas funciona de maneira inconsistente ou até gera exceções em alguns processadores. Em competições de escrita, estes opcodes acabam sendo uma modo de tornar o código menor, mais rápido ou elegante ou apenas para mostrar a profissionalidade do autor.

Sintaxe

A linguagem de montagem x86 possui dois ramais de sintaxe: sintaxe Intel, usada originalmente para documentação da plataforma x86, e sintaxe AT&T.[1] Sintaxe Intel é dominante no sistema operativo Microsoft Windows. No mundo Unix/Linux, ambas são usadas pois o GCC suportava apenas a sintaxe AT&T nos primórdios de sua criação. Aqui está um resumo das principais diferenças entre a Sintaxe Intel e a Sintaxe AT&T:

Registros

Processadores x86 possuem uma coleção de registros disponíveis para serem usados como armazéns para dados binários. Coletivamente os dados e os endereços de registro são chamados de registros gerais. Cada registro possui um propósito especial além do que eles todos podem fazer.

- AX multiplicar/dividir
- BX registro indicador para MOVE
- CX registro de contagem para operadores de linha
- DX endereço de porta para IN e OUT

Além dos registro gerais, adicionalmente há os:

- IP ponteiro de instrução
- FLAGS
- registros de segmento (CS, DS, ES, FS, GS, SS) que ativam onde um segmento de 64k começa
- registros extra de extensão (MMX, 3DNow!, SSE, etc.).

O registro IP aponta para onde no programa o processador está atualmente executando o seu código. O registro IP não pode ser acessado pelo programador diretamente.

Endereço segmentado

A arquitetura x86 em modo real e virtual 8086, usa um processo conhecido como segmentação de endereços de memória, não o modelo de memória plana, usado em muitos outros ambientes. Segmentação envolve a composição de um endereço de memória de duas partes, um segmento e um deslocamento; pontos do segmento até o início de um grupo de 64 KB de endereços, e o deslocamento determina o quão longe desse endereço iniciar o endereço desejado. No endereçamento segmentado, dois registros são necessários para um endereço completo da memória: um para segurar o segmento, a outra para realizar o deslocamento. A fim de traduzir de volta para um endereço de plano, o valor de segmento é deslocado quatro bits à esquerda (o equivalente a multiplicação por 24 ou 16), em seguida, adicionado ao deslocamento para formar o endereço completo, que permite quebrar a barreira de 64k com a escolha inteligente de endereços, endereçar até 1 MB de memória enquanto estiver usando 16 bits, ao invés de registros de 32 bits, embora ele torna a programação muito mais complexa. No modo real, apenas, por exemplo, se DS contém o número hexadecimal 0xDEAD, e DX contém o número 0xCAFE, eles juntos apontariam para o endereço de memória 0xDEAD * + 0x10 = 0xCAFE 0xEB5CE. No modo real o processador pode endereçar até 1.048.576 bytes (1 MB). Isto aplica-se a 20-bits espaço de endereço. Ao combinar valores de deslocamento do segmento encontramos um endereço de 20 bits. O PC original de IBM restringiu programas a 640 KB, mas uma especificação da memória expandida foi usada para executar um esquema do interruptor de banco que caísse fora do uso quando uns sistemas operacionais posteriores, tais como Windows, usaram as escalas de endereço maiores de uns processadores mais novos e executaram seus próprios esquemas da memória virtual. O Modo Protegido, começando com o processador Intel 80286, foi utilizado pelo OS / 2. Várias deficiências, como a impossibilidade de acessar o BIOS, e da incapacidade de voltar ao modo real sem precisar reiniciar o processador. impediram uma utilização generalizada. O 80286 também estava limitado a endereçamento de memória em segmentos de 16 bits, ou seja, apenas 216 bytes (64 kilobytes) podiam ser acessados ao mesmo tempo. Para acessar a funcionalidade estendida do 80286, o sistema operacional deveria definir o processador em modo protegido, permitindo endereçamento de 24 bits e, portanto, 224 bytes de memória (16 megabytes). Ao se referir a um endereço com um segmento e um deslocamento, a notação de segmento: offset é usada, assim, no exemplo acima, o endereço 0xEB5CE plano pode ser escrito como 0xDEAD: 0xCAFE ou como um segmento, e o deslocamento par de registradores; DS: DX.

Registradores de Segmento:

CS: Segmento de Código;
DS: Segmento de Dados;
SS: Segmento de Pilha;
ES: Segmento de Dados Extra;
FS: Segmento de Dados Extra 2;
GS:Segmento de Dados Extra 3;

Registradores de Ponteiros e Índice, usados para armazenar endereços de memória:

ESP: Ponteiro de pilha
EBP: Ponteiro de base da pilha
ESI: Ponteiro de índice fonte (pode servir como registradores de uso geral)
EDI: Ponteiro de índice destino (pode servir como registradores de uso geral)
EIP: Ponteiro de instrução (contador de programa)

O Intel 80386, apresentou três modos de operação: modo real, modo protegido, e modo virtual. O modo protegido, que estreou no 80.286 foi estendido para permitir o 80386 para endereçar até 4 GB de memória, o novo modo 8086 virtual, (vm86) tornou possível executar um ou mais programas de modo real em um ambiente protegido, que em grande parte imitando modo real, embora alguns programas não eram compatíveis (geralmente como resultado de endereçamento de memória usando truques ou indeterminado op-codes). O modelo de memória plana de 32 bits, do modo protegido, estendida até 80386 pode ser a mudança mais importante da característica para a família de processadores x86, até que a AMD lançou x86-64 em 2003, ajudou a impulsionar a adoção em grande escala do Windows 3.1 (que se baseava em modo protegido), desde que o Windows pode agora executar vários aplicativos simultaneamente, incluindo aplicativos DOS, usando memória virtual e multitarefa simples.

Modo de Execução

Os processadores x86 sustentam cinco modos de operação para o código 86, o modo real, o modo protegido, modo longo, modo 86 virtual, e administração de sistemas, em que algumas instruções estão disponíveis e outras não. Um subgrupo de 16 bits de instruções estão disponíveis no modo real (todos os processadores x86), o modo 16-bit protegido (80286 em diante), V86 (80386 e posteriores) e CEM (Alguns i386SL Intel, i486 e posteriores). Em 32 bits do modo protegido (Intel 80386 em diante), instruções de 32 bits (incluindo as extensões mais tarde) também estão disponíveis, no modo de tempo (a partir de AMD Opteron), instruções de 64 bits, e registra mais, também estão disponíveis. O conjunto de instruções é semelhante em cada modalidade, mas de endereçamento de memória e tamanho da palavra variam, exigindo estratégias de programação diferentes.

Os modos em que o código x86 pode ser executado são:

Modo real (16 bits): Modo Real é caracterizado por um espaço de endereço de 20 bits de memória segmentada, (dando um pouco mais de 1 MB de memória endereçável) e software de acesso ilimitado e direto da memória e endereços de I / O e hardware periférico. Modo Real não oferece suporte para proteção de memória, multitarefa, ou níveis de privilégio de código.

O modo protegido (16 bits e 32 bits): Em computação, de modo protegido, também chamado de modo de endereço virtual protegido, é um modo operacional de x86 compatível com unidades de processamento central (CPU). Ele permite que o software do sistema utilize recursos como memória virtual, paginação, multi-tarefa segura e outros recursos projetados para aumentar o controle de um sistema operacional em relação ao software aplicativo.

Modo longo (64 bits): É a modalidade onde uma aplicação 64 – bit (ou o sistema de exploração), podem alcançar as instruções e os registros de 64-bit. Os programas de 32

bits e os programas de 16 bits do modo protegido são executados em um sub-modo da compatibilidade; o modo real ou o modo virtual 8086 não podem funcionar nesta modalidade.

Modo Virtual 8086 (16 bits): Permite a execução de aplicativos de modo real que são incapazes de rodar diretamente em modo protegido, enquanto o processador está executando um sistema operacional de modo protegido.

Modo de administração de sistemas (16 bits): É um modo em que toda a execução normal (incluindo o sistema operacional) é suspenso. Foi lançado com a Intel 386SL

Alternando entre modos

O processador entra em modo real imediatamente após ligar, assim que um kernel de sistema operacional, ou outro programa, deve explicitamente alternar para outro modo, se pretende executar alguma coisa, mas de modo real. Modos de comutação são feitos modificando certas partes dos registos do processador de controle, embora alguns requerem preparação prévia, em muitos casos, e alguma limpeza no interruptor pode ser necessária.

Em geral, as características do moderno conjunto de instruções x86 são:

- A codificação compacta
- Tamanho variável e alinhamento são independentes
- Principalmente nas instruções de um e dois endereços, o primeiro operando é também o destino.
- Operandos da memória como o de origem e de destino são suportados
- Uso geral e implícito do registro
- Produz bandeiras condicionais implicitamente com a maioria de instruções da ULA.
- Suporta vários modos de endereçamento incluindo imediato, deslocamento e índice escalado, mas não relativo ao PC, com exceção de saltos (apresentado como uma melhoria na arquitetura x86).
- Inclui em ponto flutuante para uma pilha de registos.
- Contém a sustentação especial para instruções atômicas
- Instruções SIMD

Instruções de pilha

A arquitetura x86 tem suporte de hardware para a execução do mecanismo da pilha. Instruções como o push, pop, call e ret são usados corretamente com a pilha para passar parâmetros, alocar espaço para dados locais e para salvar e restaurar pontos de retorno de chamada. A instrução ret size é muito útil para a aplicação eficiente em termos de espaço nas convenções de chamada, onde o receptor é responsável pela recuperação do espaço da pilha ocupado por parâmetros.

Ao configurar um quadro de pilha para armazenar dados locais de um procedimento recursivo, há várias opções, a instrução enter de alto nível leva um argumento de procedure-nesting-depth, bem como um argumento de tamanho local, e pode ser mais rápida do que a manipulação explícita da registradores, mas geralmente não é usado. Se

ele é mais rápido depende da aplicação particular x86, bem como a convenção de chamada e o código que se deseja executar em vários processadores normalmente irá correr mais rápido na maioria dos alvos sem ele.

A série completa de modos de endereçamento, mesmo para obter instruções sobre push e pop, faz uso direto da pilha de inteiros, ponto flutuante e os dados de endereço simples, bem como mantimento das especificações ABI e os mecanismos relativamente simples comparado a algumas arquiteturas RISC.

Instruções de Ponto Flutuante

A linguagem assembly x86, inclui instruções para uma unidade de ponto flutuante baseado em pilha. Eles incluem a adição, subtração, negação, multiplicação, divisão, resto de divisão, raízes quadradas, o truncamento inteiro, o truncamento da fração, e de escala por potência de dois. As operações também incluem instruções de conversão que podem carregar ou armazenar um valor de memória em qualquer um dos seguintes formatos: decimal codificado binário, inteiro de 32 bits, inteiro de 64 bits, 32 bits de ponto flutuante, ou ponto flutuante de 80 bits (após o carregamento, o valor é convertido para o modo de ponto flutuante usado atualmente). X86 também inclui uma série de funções transcendentais incluindo seno, cosseno, tangente, exponenciação, e logaritmos com bases 2, 10 ou hexadecimal. Como nos números inteiros, o primeiro operando é tanto o operando fonte como operando destino, fsubr fdivr devem ser apontados como o primeiro, trocando os operandos fonte antes de executar a subtração ou divisão. As instruções de adição, subtração, multiplicação, divisão, armazenar e comparação incluem modos de instrução que irá aparecer no topo da pilha após a operação ser concluída.

Instruções SIMD

CPUs x86 modernas contêm instruções SIMD, que em grande parte, executam a mesma operação em paralelo em muitos valores codificados num registro SIMD de largura. Várias tecnologias de instruções apoiam as operações em diferentes conjuntos de registros diferentes, mas tomadas em conjunto completo, que incluem cálculos gerais sobre inteiro ou aritmética de ponto flutuante (adição, subtração, multiplicação, deslocamento, minimização, maximização de comparação, divisão ou raiz quadrada). SSE também inclui um modo de ponto flutuante em que apenas o primeiro valor dos registros é realmente modificado (ampliado em SSE2). Algumas outras instruções incomuns foram adicionadas, incluindo uma soma das diferenças absolutas (utilizados para a estimativa de movimento em compressão de vídeo, como é feito em MPEG) e uma de 16 bits multiplica instrução acumulada. SSE (desde SSE3) e 3DNow! extensões que incluem adição e subtração de instruções para o tratamento, combinando valores de ponto flutuante como números complexos. Estes conjuntos de instruções também incluem numerosas instruções sub-palavra para baralhar, inserindo e extraindo os valores ao redor e dentro dos registros. Além disso, há instruções para mover dados entre os registros inteiro e MMX.

Instruções de Manipulação de Dados

O processador x86, também inclui modos de endereçamento complexos para lidar com a memória com um imediato deslocamento, um registro, um registro com um

deslocamento, registrar uma escala com ou sem deslocamento, e um registro com um opcional offset, e, outro registro escalado. Assim, por exemplo, um pode codificar mov eax [Table + ebx + esi*4], como uma única instrução que carrega 32 bits de dados a partir do endereço computado como: (Table + ebx + esi * 4) deslocado do seletor de ds, e armazená-lo para o registro eax. Em processadores x86 pode carregar e usar a memória compatível com o tamanho de qualquer registro que se operar. (As instruções SIMD também incluem instruções de meia carga.) O conjunto de instruções x86 inclui carreagar a string, e instruções de movimento (LODs, OCP e movs) que executam cada operação para um determinado tamanho (b para byte de 8 bits, w para a palavra de 16 bits, d para 32bits palavra dupla), então incrementos / decrementos (dependendo do DF, flag de direção) ocorrem no registo de endereço implícito (SI para LODs, di para stos e tanto para movs). Para carregar o registro implícito do alvo/fonte está no registro al, ax ou eax (dependendo do tamanho). O segmento implícito usado é ds para lods, es para stos e ambos para movs. A pilha é implementada implicitamente com um decremento (push) e incremento (pop) ponteiro de pilha. No modo de 16 bits, esse ponteiro implícito da pilha é tratado como SS: [SP], no modo de 32 bits é SS: [ESP], e no modo de 64 bits é [RSP]. O ponteiro da pilha realmente aponta para o último valor que foi armazenado, sob a suposição de que o seu tamanho vai coincidir com o modo de operação do processador (ou seja, 16, 32 ou 64 bits) para coincidir com a largura padrão do push / pop / chamada / instruções ret. Também estão incluídas as instruções que reservam e retiram dados do topo da pilha, enquanto a criação de um ponteiro da estrutura pilha em pb / ebp / ead. No entanto a configuração, direta ou adição e subtração para o sp / ESP / registro rsp também é suportada, portanto, o entrar / sair instruções são geralmente desnecessários.

Código do início da função:

```
  pushl   %ebp   / * Função de salvar quadro de pilha de chamada (% ebp)
  movl      %esp,%ebp /* Fazer um novo quadro de pilha em cima da pilha * nosso interlocutor /
  subl    $4,%esp  / * Atribuir 4 bytes de espaço de pilha para * esta função de variáveis locais é funcionalmente equivalente a: enter $4,$0
```

Outras instruções para manipular a pilha incluem pushf / popf para armazenar e recuperar os Flags. As instruções pusha/ popa vão armazenar e recuperar o número inteiro de registro de estado de/ para a pilha. Valores para umSIMD carregar ou armazenar são assumidos para serem embalados em posições adjacentes do registo SIMD e alinhá-las em ordem sequencial little-endian. Alguns carregam instruções SSE e para armazenar exigem alinhamento de 16 bytes para funcionar corretamente. Os conjuntos de instrução SIMD também incluem "prefetch" instruções que realizam o carregamento, mas não são alvo de qualquer registro, utilizadas para o carregamento do cache. Os conjuntos de instruções SSE também incluem instruções de armazenamento não-temporal que irão realizar os armazenamentos direto para a memória sem a realização de um cache se o destino já não está em cache (caso contrário, ele irá se comportar como um armazenamento normal.)

A maioria das instruções genéricas inteiras e de ponto flutuante (mas não SIMD) podem usar um parâmetro de como um endereço complexo como o parâmetro da segunda fonte. instruções inteiras podem também aceitar um parâmetro de memória como operando destino

Fluxo de Programa

O conjunto x86 tem uma operação do salto incondicional, o jmp, que pode tomar um endereço imediato, um registro ou um endereço indireto como um parâmetro (note que a maioria dos processadores RISC suportam apenas um registo link ou deslocamento imediato curto para pular). Também são suportados vários saltos condicionais, incluindo JE, JNE, JG, JL, JA JB.

- JE: desvia se igual;
- JNE: desvia se diferente;
- JZ: se o ZF for igual a 1 (ZF= zero flag);
- JMP: desvio incondicional;

Estas operações condicionais são baseadas no estado de bits específicos em registradores Flags. Muitas operações aritméticas e lógicas estabelecidas, claras ou complementar estes Flags, dependendo o seu resultado. A comparação cmp (compare) e instruções de teste definir os sinalizadores, como se tivessem realizado uma subtração ou um bit a bit E operação, respectivamente, sem alterar os valores dos operandos. Há também instruções de como clc (flag de carry claro) e CMC (complemento de bandeira) que trabalham diretamente nas bandeiras. comparações de ponto flutuante são executadas através FCOM Ficom ou instruções que eventualmente tem que ser convertido para inteiro bandeiras.

Assembly

Conjunto de instruções são as operações que um processador, microprocessador, CPU ou outros periféricos programáveis suporta, fornece ou disponibiliza para o programador, ou seja, é a representação em mnemônicos do código de máquina, com a finalidade de facilitar o acesso ao componente.

Cada componente possui o seu próprio conjunto de instruções, que é fornecido pelo fabricante, que também costuma fornecer ou disponibilizar um montador assembly, que transforma o conjunto de instruções em código de máquina para ser utilizado pelo componente.

No caso dos processadores, quando o conjunto de instruções for reduzido leva-o a ter o nome de RISC e se forem complexas o nome de CISC.

Exemplos de códigos assembly em processadores ARM:

As instruções de processamento de dados geralmente tem 3 argumentos:

- op1: O primeiro argumento é um registrador onde o resultado é armazenado
- op2: O segundo argumento é um registrador usado como primeiro operando da operação
- op3: O terceiro argumento é o segundo operando da operação, que pode ser um valor imediato de 8 bits, um registrador ou um registrador com deslocamento.

As instruções de 3 argumentos são as seguintes:

instruções de 3 argumentos		
AND op1,op2,op3	op1 = op2 & op3	E bit a bit
EOR op1,op2,op3	op1 = op2 ^ op3	Ou exclusivo
ADD op1,op2,op3	op1 = op2 + op3	Soma
SUB op1,op2,op3	op1 = op2 - op3	Subtração
RSB op1,op2,op3	op1 = op3 - op2	Subtração reversa
ADC op1,op2,op3	op1 = op2 + op3	Soma com carry
SBC op1,op2,op3	op1 = op2 - op3	Subtração com carry
RSC op1,op2,op3	op1 = op3 - op2	Subtração reversa com carry
ORR op1,op2,op3	op1 = op2 | op3	Ou bit a bit
BIC op1,op2,op3	op1 = op2 & ~op3	Bit clear (zera bits)

Exemplos de códigos assembly em processadores X86:

Instruções aritméticas usam dois operandos: um destino e uma fonte. O destino deve ser um registo ou um local de memória. A fonte pode ser, quer uma posição de memória, um registo, ou um valor constante. Note-se que pelo menos um dos dois deve ser um registo, porque as operações podem não utilizar uma localização de memória tanto como uma fonte e um destino.

add src, dest GAS Syntax

add dest, src Intel syntax

Isso adiciona src para dest. Se você estiver usando a sintaxe MASM, então o resultado é armazenado no primeiro argumento, se você estiver usando a sintaxe GAS, ele é armazenado no segundo argumento.

sub src, dest GAS Syntax

sub dest, src Intel syntax

Como add, só que subtrai fonte de destino, em vez de adionar.

ARM versus X86

Algumas décadas se passaram desde o lançamento da primeira CPU para consumidores. Nesse meio tempo, processadores antigos foram aposentados e novos modelos se apresentaram. A batalha entre fabricantes aumentou ainda mais, contudo, o foco agora

não é mais a briga entre Intel e AMD. Novas empresas surgiram para apostar em um novo competidor: o ARM.

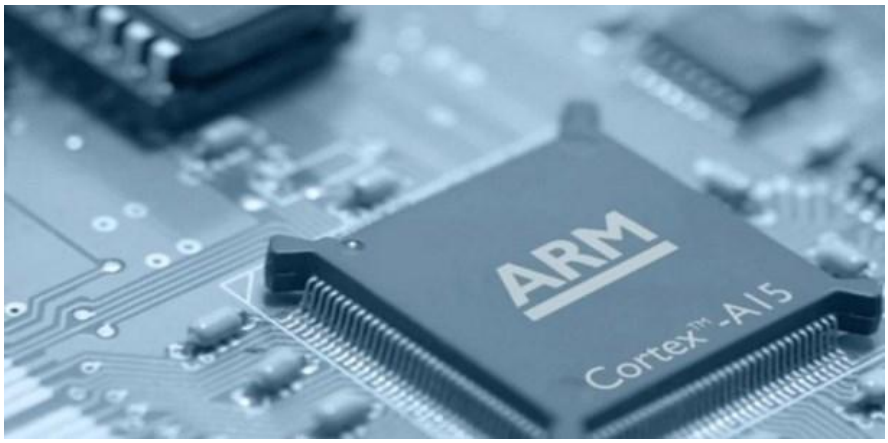
Os responsáveis pela explosão de sucesso do ARM são os smartphones e os tablets. Com isso, algumas corporações estão suscitando a ideia de que a novidade pode ser interessante na substituição da atual arquitetura que reina no mercado dos desktops.

Nosso primeiro combatente é o x86. Em um primeiro instante, a arquitetura x86 era exclusiva da Intel. A fabricante aproveitou essa arquitetura em seus primeiros processadores, incluindo os famosos 80386 e 80486 — conhecidos apenas pelas abreviações 386 e 486. A terminologia “86” então foi aproveitada para criar uma arquitetura genérica, conhecida como x86.

*O início da arquitetura x86 (Fonte da imagem: **Reprodução/Konstantin Lanzet**)*

Ela foi usada posteriormente em CPUs da AMD, Cyrix, VIA e da própria Intel. Atualmente, o termo é utilizado para designar qualquer processador compatível com instruções de 32 bits. Na lista de modelos compatíveis com a arquitetura X86, também estão presentes todos os modelos de 64 bits, porém, o mercado diferencia tais modelos utilizando a sigla x64.

Do outro lado do ringue, está o ARM. A arquitetura ARM nasceu alguns anos depois da x86, mas foi ofuscada durante longas décadas por não estar presente nos dispositivos que atingiam a grande população. Desenvolvida pela ARM Holdings, essa arquitetura veio para simplificar as instruções de 32 bits.



*A evolução notável do ARM (Fonte da imagem: **Divulgação/ARM**)*

Apesar de um longo tempo de existência, a ARM só veio brilhar recentemente, com o aparecimento dos smartphones e tablets. Hoje, empresas como Samsung, Texas Instruments, Qualcomm e NVIDIA estão investindo alto no desenvolvimento dessa arquitetura. As fabricantes de aparelhos portáteis também entram em campo, com dispositivos cada vez mais potentes e repletos de recursos — graças aos chips ARM e outras tecnologias.

Round 1: energia

O consumo de energia dos processadores nunca foi um fator tão importante no ramo dos desktops. Já para smartphones e tablets, economizar bateria é fundamental. Com isso temos a primeira batalha entre as arquiteturas.

No segmento de portáteis, o consumo de energia sempre foi o grande enfoque. As fabricantes de processadores sempre tiveram em mente que o desempenho não é o mais importante, mas sim a duração da bateria. Com isso, a arquitetura ARM sempre primou por utilizar um mínimo de recursos para garantir a utilização do aparelho por um tempo prolongado.

Com o lançamento dos mais recentes net e notebooks, tanto Intel como AMD surpreenderam com modelos de baixo consumo de energia. O Intel Atom e o AMD eBrazos são exemplos claros de que é possível criar processadores x86 de baixo consumo, mantendo o alto desempenho em laptops modestos.

Round 2: velocidade

A arquitetura x86 tem frequências absurdamente altas. As CPUs baseadas na arquitetura ARM ainda não tiveram tanto tempo para exigir clocks na mesma proporção.

Por se tratar de um mercado voltado a diversos tipos de usuários, os desktops sempre precisaram de mais poder de processamento. Isso justifica os atuais modelos que têm

núcleos trabalhando a mais de 3 GHz. Existem ainda os processadores x86 mais básicos, os quais trabalham com clocks próximos a 1,6 GHz.

Os atuais processadores ARM trabalham com frequências próximas a 1,2 GHz (alguns superam essa marca). E as promessas das fabricantes são para o aumento do clock acima dos 2 GHz — como é o caso do processador top de linha Qualcomm Snapdragon, que deve operar na frequência de 2,5 GHz.

Round 3: aquecimento

Uma grande preocupação com CPUs baseadas na arquitetura x86 sempre foi o superaquecimento. Com os novos processadores ARM, o problema parece ter sido resolvido.

Por trabalharem em frequências menores e consumirem pouca energia, a geração de calor das CPUs ARM é mínima. Tal fato é comprovado nos diversos smartphones e tablets que dispensam o uso de coolers.

Os modelos top de linha para desktops e notebooks esquentam muito. Consequentemente utilizam sistemas de refrigeração robustos, compostos por dissipador e cooler. No entanto, os novos processadores x86 para netbooks, por exemplo, exigem sistemas mais básicos, visto que esquentam muito menos.

Round 4: tecnologia

Com a evolução dos processadores x86, diversas novas tecnologias chegaram para possibilitar a execução aprimorada de determinados aplicativos. Nos processadores ARM, algumas tecnologias ainda não estão presentes, mas a aposta é de que os aplicativos se adaptem à nova arquitetura.

Quem pensa em executar o Adobe Photoshop, o AutoCAD e outros tantos softwares específicos não tem como pensar em outra solução que não seja um computador comum. Os processadores baseados na arquitetura x86 trazem instruções complementares (como a SSE, a AVX e outras), as quais estão presentes apenas neles e possibilitam a execução de funções únicas.

É fato que os processadores ARM atuais não têm capacidade para executar os mesmo aplicativos criados para CPUs x86. Contudo, as fabricantes vêm apostando que as desenvolvedoras de programas irão trabalhar com alternativas para possibilitar a execução de tais softwares. Se isso vai acontecer, ninguém sabe, mas é certo que se o ARM migrar para os desktops, as chances aumentam.

Round 5: crescimento de mercado

Os processadores ARM estão impulsionando a venda de diversos produtos. Já as CPUs x86 continuam com uma fatia grande de mercado.

Não há como negar, a arquitetura ARM está crescendo em um ritmo acelerado. Antes, eles eram incomuns, agora são essenciais para considerar um dispositivo como “smart”. A Sony, por exemplo, vai adotar uma CPU ARM no PlayStation Vita. E é devido a esse “boom” repentino que muitas companhias estão apostando na migração do ARM para o mercado de desktops e notebooks.

Apesar de as CPUs ARM terem chegado para ficar, a arquitetura x86 não perdeu quase nada de mercado. As duas coexistem e, até o momento, a arquitetura ARM não roubou o mercado de desktops. Além disso, devemos lembrar que as CPUS x86 já concorriam com outras arquiteturas, como as da IBM, por exemplo, que dominam nos consoles atuais.

Round 6: sistemas operacionais

A enorme gama de sistemas deve dificultar um pouco a propagação da arquitetura ARM. Os processadores x86 não parecem ter esse problema, visto que eles não estão tentando dominar outro mercado.

Verdade seja dita, nenhum processador ARM consegue executar o Windows 7 ou o Mac OS X Lioncom perfeição, até porque esses sistemas não trazem compatibilidade para tais CPUs. E não é só isso: a maioria dos sistemas operacionais foca no desenvolvimento para a arquitetura x86.

A Microsoft já anunciou que o Windows 8 trará suporte para a arquitetura ARM. Enquanto isso, o Linux já tem suporte para esses processadores. A dúvida que permanece é se o desempenho dos ARMs fará frente ao dos modelos x86. Segundo rumores, a Apple pode estar pensando em mudar os processadores de seus notebooks para a arquitetura ARM. Isso sim faria a arquitetura ARM tornar-se prioritária. Resta esperar para ver.

Round 7: aposta de grandes empresas

Vá à loja de uma operadora telefônica e confira quantas marcas de smartphone existem. As empresas de telefonia são apenas algumas das tantas que apostam no ARM. Por outro lado, a venda de notebooks está em alta, o que significa que o x86 tem muita força ainda.

Em síntese, muito se fala que os ARM estão dominando nos tablets e smartphones. Contudo, por trás dos panos, está claro que eles estão sendo instalados em televisores, geladeiras e outros produtos. Ao que tudo indica, as empresas pretendem fazer a

interação entre todos (ou quase) os dispositivos eletrônicos. O ARM talvez seja a solução.

Apesar de o ARM almejar o futuro, a arquitetura x86 tem aliados de peso, incluindo Dell, HP, Apple e todas as demais montadoras de computadores.

Round 8: quantidade de núcleos

Apesar de os propósitos das arquiteturas serem diferentes, a quantidade de núcleos é um fator que faz toda a diferença. Considerando ainda que a arquitetura ARM talvez domine o mercado de desktops, esse fator deve ser levado em conta.

Ocorreu, nessa segunda-feira (25), o vazamento de um cronograma revelando os modelos que, teoricamente, serão os grandes trunfos da AMD para 2012. Segundo o documento, a fabricante vai lançar modelos para desktops com até dez núcleos. Tais processadores devem operar em alta frequência e contar com placa gráfica Radeon HD 7000.

Durante a MWC 2011, a Qualcomm revelou que sua próxima geração de CPUs ARM trará até quatro núcleos operando a 2,5 GHz. As demais fabricantes de processadores desse tipo também apostam alto, com modelos semelhantes.

Conclusão

Assembly ou linguagem de montagem é uma notação legível por humanos para o código de máquina que uma arquitetura de computador específica usa. A linguagem de máquina, que é um mero padrão de bits, torna-se legível pela substituição dos valores em bruto por símbolos chamados mnemônicos. É utilizada em geral para programação em baixo nível de máquina (mais próxima do nível do *hardware*), sendo que cada família de processadores (Ex. X86, ARM, SPARC, MIPS) possui sua própria linguagem assembly, já que cada processador possui seu próprio conjunto de instruções embutidas.

Muitos dispositivos programáveis (como microcontroladores) ainda são programados diretamente em assembly apesar de que a partir da década de 1970 muitos dispositivos passaram a ser programados em C.

A linguagem Assembly, é intimamente ligada ao processador da CPU, dependendo inteiramente do *hardware*: cada processador tem, assim, sua própria linguagem Assembly (Ex. X86, ARM, SPARC, MIPS).

Vantagens

- Velocidade, os programas em Assembly são, em geral, mais rápidos;
- Tamanho dos programas de Assembly são menores;
- Assembly permite criar ações de alta complexidade, impossíveis ou difíceis de se realizar em linguagens de Alto Nível;
- Conhecimento, em Assembly possibilita a programação nos outros tipos de linguagem;
- É um programa recomendável onde o tempo é um fator crítico como por exemplo: medições de tempo que exigem boa performance;
- Se torna de fácil compreensão com algum conhecimento de conceitos de hardware e seus dialetos.

Desvantagens

- Difícil de ler, de aprender, entender, debugar e difícil manutenção. Um programa escrito em linguagem Assembly não é muito legível, por isso ele deve ser muito bem documentado.
- Programar em Assembly consome muito tempo para o programador;
- A linguagem não é portável. Ela é portável apenas dentro de uma família de processadores.
- Como é uma linguagem especifica para processares de cada máquina, é necessário desenvolver um programa para cada máquina.
- Não existe rotinas pré-definidas, o programador deverá desenvolver suas próprias rotinas.
- A linguagem Assembly apresenta um número muito reduzido de instruções, do tipo operações de movimentação de dados em memória, para registros e para memórias, e operações lógicas e aritméticas bem simples. Estas instruções são de baixa expressividade, isto é, elas são de baixo nível. O programador deve programar num nível de detalhamento muito maior para fazer a mesma coisa que em um programa escrito em linguagem de alto nível.

- Como o programador utiliza diretamente os recursos do processador e memória, ele deve conhecer muito bem a máquina onde ele está programando.

Referencias

https://pt.wikipedia.org/wiki/RISC

https://pt.wikipedia.org/wiki/CISC

https://pt.wikipedia.org/wiki/Arquitetura_ARM

https://pt.wikipedia.org/wiki/Conjunto_de_instru%C3%A7%C3%B5es

https://pt.wikiversity.org/wiki/Introdu%C3%A7%C3%A3o_%C3%A0s_Linguagens_de_Programa%C3%A7%C3%A3o/Assembly

https://en.wikibooks.org/wiki/X86_Assembly/Arithmetic

https://pt.wikipedia.org/wiki/X86

https://pt.wikipedia.org/wiki/Linguagem_de_montagem_x86