Uma visão geral da arquitetura x86-64 e de processadores que a implementam

Alan Kleiman
RA 041438

ABSTRACT

O propósito desse artigo é oferecer uma visão geral das mudanças introduzidas pela ISA x86-64, utilizado nos processadores Athlon 64 e Pentium 4 com EMT64. Entre as mudanças nessa arquitetura estão alterações no número de registradores, que passa a ser 16 registradores de uso geral. Além disso, o número de registradores XMM (utilizados em instruções SSE) também aumentou de 8 para 16. Os registradores de uso geral passam a ter um comprimento de 64 bits. O mecanismo de endereçamento também mudou, permitindo um espaço de endereçamento de 64 bits, plano (sem a segmentação presente na ISA x86). Além disso, agora é possível endereçar a memória a partir do RIP (ponteiro de instrução). Outra mudança que vale mencionar foi a adoção das instruções SSE e SSE2 como padrão na arquitetura, na prática tornando outros mecanismos de ponto flutuante (MMX, x87 FPU, 3DNow!) obsoletos, e aproximando o tratamento de ponto flutuante ao encontrado em arquiteturas RISC.

1. INTRODUÇÃO

A ISA (*instruction set architecture*) x86-64, também conhecido como AMD64, EMT64 e x64 consiste de uma extensão e limpeza da ISA IA32, introduzida pela Intel com o processador 386. Como o próprio nome indica, ele foi expandido para trabalhar com dados de 64 bits de largura e endereçar 64 bits de memória (embora implementações atuais do processador sejam mais limitadas que isso). Além disso, o número de registradores foi aumentado: antes eram 8 registradores de uso geral (GPR) e 8 pra instruções *Streaming SIMD* (XMM). Agora são 16 registradores de uso geral e 16 XMM.

Mas além dessa expansão foram feitas várias limpezas de modo a remover várias das idiossincrasias presentes na ISA IA32; contanto, devido à necessidade de se manter a compatibilidade essas mudanças apenas estão presentes quando o processador opera em modo 64 bits.

Dentre essas mudanças está o modelo de memória "achatado"[1]. Esse foi implementado devido ao fato de que a maior parte da segmentação do espaço de endereçamento em sistemas modernos é feito em software, pelo sistema operacional. Então a memória é acessada como um bloco contínuo de 64 bits, ao contrário do que havia nos processadores anteriores, em que se mantinha um apontador para segmentos de código e dados.

Outra alteração importante é que instruções das famílias SSE (*Streaming SIMD Extensions* 1/2/3) foram inclusas na especificação do processador; devido a esse fato a Microsoft, no Windows XP x86-64 dá preferência às instruções SSE sobre as instruções de outras extensões, como MMX, 3dNow! e o próprio FPU x87. O sistema operacional simplesmente não salva os dados contidos nesses registradores em mudanças de contexto.

Além disso, a arquitetura do x86-64 especifica o bit **NX** (No eXecute) como um mecanismo para prevenir erros de *buffer overflow*. Consiste de um bit na tabela de páginas que especifica se os dados em dada tabela podem ou não serem executados.

Um comentário: esse artigo não vai tratar de mudanças no *hardware* em si, e sim focar no conjunto de instruções e arquitetura associada desenvolvida pela AMD e depois adotada pela Intel, a ISA x86-64. Portanto detalhes como o gerenciamento de memória embutida do Athlon 64 ou o aumento do número de estágios de *pipeline* adotados nessa implementação não receberão atenção.

2. MODOS DE EXECUÇÃO

A arquitetura provê dois modos principais de execução: *Long mode* e modo legado. O long mode é o modo de execução pretendido para o processador; é nele que executa o código de 64 bits. Esse modo tem dois submodos: modo de compatibilidade, no qual código de 32 bits pode ser executado sem nenhuma perda de desempenho, e modo 64 bits, no qual executa código de 64 bits, como o próprio nome indica. Vale notar que os outros modos de execução (modo virtual e real) não são modos válidos, enquanto o processador está em long mode — apenas código do modo protegido pode ser executado no modo de compatibilidade.

O outro modo de execução é o modo legado; ele executa como um processador x86, inclusive permitindo executar

[1] *Flat.*, em inglês

código em modo virtual e real.

Referimos a [2] para detalhes de como cada modo de execução é acionado. A tabela 2 ilustra os diferentes modos.

Quase todas as mudanças mencionadas nesse artigo se referem ao modo 64 bits, uma vez que os outros modos são idênticos, no nível da ISA, aos modos presentes nos processadores x86-32. Os long mode pode ser acessado apenas por um sistema operacional de 64 bits, enquanto que o modo legado é pretendido para sistemas operacionais mais antigos e programas que precisam executar em 8 ou 16 bits.

Table 1: Modos de operação do x86-64

Modo de operação		Sistema operacional	Endereçamento	Operando Padrão
Modo	**Submodo**			
Long	64-bit	64-bit	64-bit	32-bit
Long	Compatibilidade	64-bit	32/16-bit	32/16-bits
Legado	Protegido	32-bit	32/16-bit	32/16-bit
Legado	Virtual	32-bit	16-bit	16-bit
Legado	Real	16-bit	16-bit	16-bit

3. REGISTRADORES

Em modo de 64 bits a ISA x86-64 introduz 16 novos registradores: 8 GPRs, que deixam o número total de GPRs presentes no processador em 16. Os outros 8 registradores são registradores XMM, reservados para instruções Streaming SIMD, também conhecidos como SSE/SSE2/SSE3. Esse aumento de registradores ajuda a atenuar a falta de registradores presentes na arquitetura x86-32, apesar de que ainda são significativamente menos registradores do que se encontra em muitas máquinas RISC (o MIPS, por exemplo, tem 32).

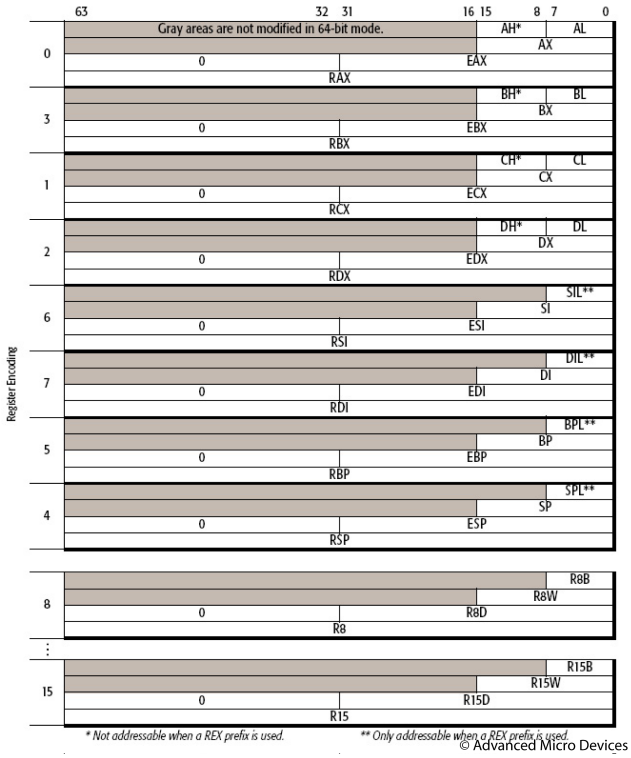
A extensão dos registradores está exibido na figura 1, retirado do *AMD64 Architecture Programmer's Manual, Volume 1.* Os registradores de uso geral de 32 bits, de EAX a ESP foram aumentados, e seus nomes, quando acessados em 64 bits vão de RAX a RSP. Além disso, foram acrescentados novos registradores, que tem nomes de R8 a R15.

Como a figura indica, registradores de tamanho menor (por exemplo, AX para os 16 bits menos significativos de EAX e AL para os 8 bits menos significativos de EAX) podem ser acessados da mesma maneira que eram na arquitetura de 32 bits. Existe uma diferença, porém, que é o prefixo REX.

O prefixo REX permite um acesso mais consistente aos registradores. Ele informa ao processador qual será o tamanho do operando de uma instrução: se não for utilizado, os mecanismos de acesso presentes na arquitetura de 32 bits são utilizados, com a diferença de que os registradores de 64 bits (Rxx) podem ser acessados. O problema desse método é que ele não é consistente. Apenas os registradores AX, BX, CX e DX podem ter os seus 8 bits mais e menos significativos acessados. Utilizando os prefixos REX podemos acessar os 8, 16, 32 ou 64 bits menos significativos de qualquer processador. Perdemos a habilidade de acessar os registradores AH a DH, mas em troca o mecanismo se torna mais consistente — o que pode ser muito importante para o desenvolvimento de um compilador eficiente.

Vale notar que na maioria dos casos as instruções tem 32 bits de largura, como padrão. Para efetuar operações sobre os registradores de 64 bits de largura é necessário utilizar um

Figure 1: Os registradores presentes na arquitetura de 64 bits



prefixo REX. Vamos ver outros detalhes sobre os prefixos REX na subseção 5.1, quando tratarmos dos formatos de instrução utilizados pelo x86-64.

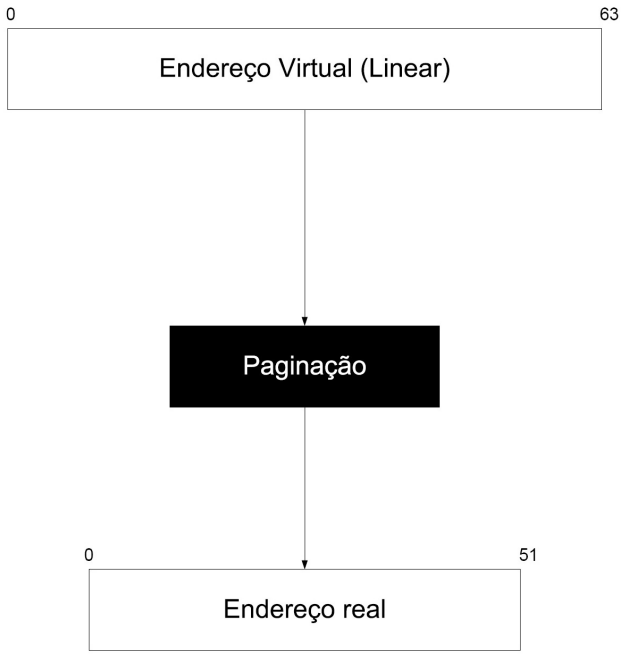
4. ENDEREÇAMENTO E ORGANIZAÇÃO DA MEMÓRIA

Como mencionado anteriormente, no modo 64-bits o modelo de endereçamento é "achatado". A memória virtual é vista como um bloco sem quebras de 64 bits, que é mapeada para o número correspondente de bits da memória real. No momento da escrita desse artigo, a memória real tinha 52 bits. O mecanismo de segmentação está ilustrado na figura 2. Embora os registradores de segmento ainda existem no modo 64-bits, o endereço contido nos mesmos é tratado como sendo 0 na maioria dos casos pra efeito de cálculo de endereçamento.

A razão dessa mudança é que em sistemas operacionais modernos o espaço de endereçamento já é visto como um bloco contíguo achatado e a segmentação é desempenhada totalmente em software. Isso permite que sistemas operacionais de 64 bits sejam codificados de uma maneira mais simples, e permite um manejamento mais simples de um ambiente multiprogramado do que era possível na arquitetura x86-32.

Embora o espaço de endereçamento é visto como um bloco contíguo, algumas características de segmentação do x86-32 ainda persistem. Ainda são utilizados segmentos de código

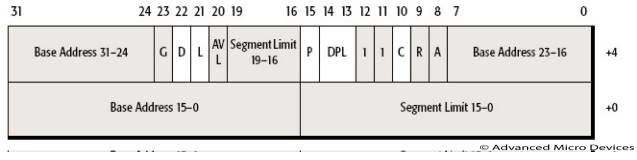
Figure 2: Paginação em modo 64-bits



e dados, embora apenas parte dos atributos de seu descritor são utilizados. Em particular, apenas os atributos que definem o privilégio, tamanhão padrão de operandos e se o código está em modo de compatibilidade ou não. Apesar de apenas usar esses atributos, o descritor completo do segmento é carregado, devido ao fato que o segmento em questão pode estar em modo de compatibilidade, ou mesmo legado.

A figura 3 ilustra quais atributos são utilizados: os atributos marcados em cinza não são utilizados. Os atributo D estabelece qual o tamanho padrão dos operandos no segmento, o atributo L estabelece se o segmento é para ser executado em long mode ou em modo de compatilidade, o atributo P estabelece se aquele segmento está presente em memória, o DPL estabelece a prioridade daquele segmento e o atributo C identifica a conformidade daquele segmento. Para mais detalhes sobre esses atributos, referimos novamente à [2]. Já em segmentos de dados apenas o atributo P é lido em modo 64-bits, todos os outros são ignorados.

Figure 3: Descritor de segmento de código em modo 64 bits.



Os registradores de segmento GS e FS, ao contrário dos outros registradores de segmento ainda podem ser utilizados como registradores base durante o endereçamento. Isso é feito utilizando uma instrução especial, que acessa um registrador específico de modelo (MSR)[2] – um registrador específico a aquela implementação do conjunto de instruções – e associa aos registradores, normalmente de 32 bits, um MSR de 64 bits com o endereço base desejado. Instruções então podem acessar endereços de memória a partir do endereço base contido nesses registradores. Tal acesso pode ser feito apenas por software com o maior privilégio, portanto não é um recurso disponível à programadores de aplicação, ao menos não diretamente. O propósito do uso desses registradores é para facilitar o acesso a dados locais em ambientes *multi-threaded*.

Uma mudança no endereçamento do conjunto de instruções x86-64 (quando em modo 64 bits) é que agora é possível acessar endereços de memória a partir do ponteiro de instrução (RIP). Antes era possível acessar dados a partir do RIP apenas em instruções de mudança de controle (*branches*, por exemplo). Agora qualquer instrução que tem o byte ModR/M (na seção seguinte veremos onde esse byte se encaixa) pode acessar dados relativos ao endereço da próxima instrução, usando o deslocamento de 32 bits. Isso foi feito para facilitar o acesso a bibliotecas compartilhadas, de modo que o *loader* não precise mais fazer alteração às referências no código durante o carregamento.

Como indica a figura 2, o mecanismo de paginação é bem simples, bem mais simples do que era o mecanismo de paginação da arquitetura x86-32, já que o mecanismo de endereçamento via segmentação não é mais utilizado.

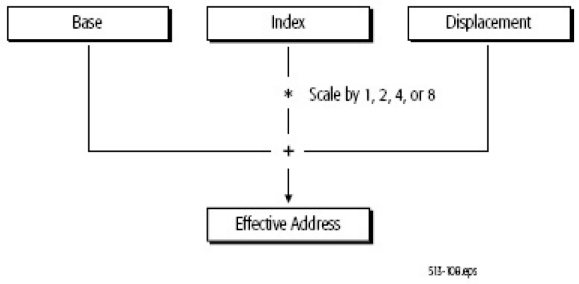
A arquitetura x86-64 exige que implementações que não conseguem endereçar todos os 64 bits do endereço virtual mantenham suas instruções em formato canônico: isso consiste em preencher os bits do bit 63 ao mais significativo implementado ou com 0s, ou com 1s. O processador verifica se os endereços estão nesse formato: se não estiverem acusa um erro. Ou um *general protection exception* ou uma falha de pilha.

A arquitetura oferece os seguintes mecanismos de endereçamento:

- **ModR/M**: Endereços utilizando endereçamento ModR/M calculam endereços utilizando uma combinação de escala, índice, base e deslocamento. A figura 4 demonstra como é feito o cálculo do endereço dadas essas informações. O índice e base são informados em bytes que seguem a instrução, enquanto que o deslocamento e a escala fazem parte do opcode. Lembrando que, na figura, na maior parte dos casos o endereço base vai ser 0, uma vez que todo o espaço de endereçamento está acessível.
- **Relativo à pilha**: Algumas instruções referenciam o apontador de pilha (RSP) implicitamente, como as instruções POP e PUSH.
- **Endereços de *strings***: Instruções que operam sobre cadeias geral endereços sequenciais implicitamente, utilizando os registradores RDI e RSI.

[2] *Model-specific register*

Figure 4: Cálculo de endereço complexo.



Instruções que contém endereços de 32 e 16 bits e estão sendo executados em long mode são expandidos para endereços de 64 bits da seguinte forma: os valores são truncados para o número correto de bits (32 ou 16) e então são extendidos com 0s até completar os 64 bits. Dessa forma, um aplicativo de 16 ou 32 bits executando em modo de compatiblidade pode apenas acessar os 4 Gigabytes mais baixos do espaço de 64 bits. Da mesma forma, um endereço de 32 bits gerado em modo 64-bits pode apenas acessar os 4 Gigabytes mais baixos.

Além disso, o maior valor para um imediato ou deslocamento na arquitetura de 64 bits é de 32 bits. Dependendo da instrução e do endereçamento, um imediato ou deslocamento pode ter 8, 16 ou 32 bits. No modo de 64-bits deslocamentos são extendidos para 64-bits, como mencionado acima. Porém para efeito de representação o máximo continua sendo 32 bits. O mesmo vale para instruções imediatas. Contanto, existem algumas exceções: alguns formatos da instrução MOV podem acessar mais do que 32 bits.

5. INSTRUÇÕES DE USO GERAL

A arquitetura x86-64 adota, na grande maioria, todas as instruções presentes na arquitetura de 32 bits, com extensões que permitem acesso a operandos e endereços de memória de 64 bits, como seria de se esperar. Contanto, algumas instruções que fazem parte da ISA x86-32 não fazem parte da arquitetura de 64 bits. Em particular, as instruções mais idiosincráticas não fazem mais parte da arquitetura, como por exemplo as instruções que lidam com valores BCD (Binary-coded Decimal) ou valores ASCII. Além disso, instruções de jump ou branch "longes" de maneira direta não são mais instruções válidas. Instruções que inserem ou removem os registradores de segmento da pilha também passam a não ser mais válidos. As instruções para chamadas de sistema também mudou: as instruções SYSENTER e SYSEXIT não são mais válidas: SYSCALL e SYSRET devem ser usadas no seu lugar.

Como mencionado anteriormente, instruções utilizam operandos de 32 bits como padrão, esse comportamento podendo ser alterado pela utilização de um prefixo REX. Já algumas instruções não precisam de um prefixo REX para operar sobre valores de 64 bits. Instruções de branch em particular operam automaticamente com valores de 64 bits; como dito na seção anterior, o valor máximo para um imediato é de 32 bits, portanto para acessar posições mais distantes é necessário fazer um acesso indireto – através de uma posição em memória.

Não vamos entrar em detalhes quanto às instruções alteradas por questões de espaço. Referimos ao manual de programação de aplicativos[1] e ao manual de instruções de uso geral e de sistema[3] para mais detalhes.

Como indicado na figura 1 operações feitas sobre valores de 32 bits são extendidos para 0; ou seja, os 32 bits mais significativos dos registradores recebem o valor 0. Isso é diferente do que acontece com instruções que manipulam os registradores de 8 ou 16 bits. Nesses casos, os bits mais significativos não são alterados pela instrução. Isso é consistente com a operação em 32 bits, na qual manipulação dos registradores de 8 e 16 bits não alterava o resto do registrador. Vale notar, inclusive, que os 32 bits mais significativos não são preservados durante mudanças de contexto, para modo de compatibilidade ou legado.

Como na arquitetura de 32 bits, vários dos registradores tem funções implícitas. Por exemplo, durante multiplicação e divisão os registradores RAX e RDX são utilizados juntos como operandos ou para armazenar os valores resultantes. O registrador RCX é utilizado em operações de laço e os registradores RBP e RSP são os registradores de pilha. Isso apesar de que os registradores são supostamente de uso geral. Não vamos mencionar cada um dos usos implícitos de registradores; invés disso referimos à [1].

O registrador EFLAGS, utilizado na arquitetura de 32 bits foi ampliado para o registrador RFLAGS, apesar de que a arquitetura x86-64 não adicionou nenhum flag novo. Os 32 bits mais significantes de RFLAGS no momento estão reservados, e o sistema garante que qualquer leitura deles, no momento, retornará 0. Novamente, não iremos entrar em detalhes e referimos a [1, 4].

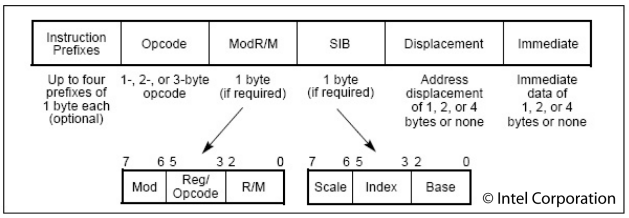
A arquitetura oferece duas instruções para chamada de código privilegiado: **SYSCALL** e **SYSRET**. As instruções contam com o fato de que o espaço de endereçamento vai ser plano, e dispensam várias checagens adicionais, além de carregarem valores pré-determinados nos segmentos apropriados, como endereços das chamadas. Essas chamadas conseguem executar em um quarto dos ciclos que levariam as instruções **CALL** e **RET**. As instruções **SYSENTER** e **SYSEXIT** também executam com baixa latência, mas são operações proibidas em modo de 64 bits.

5.1 Formato de instrução e o prefixo REX

O formato da instrução na arquitetura x86-64 é o mesmo da arquitetura de 32 bits: instruções podem ter de 1 a 15 bytes, com um prefixo de até 4 bytes, um opcode de até 3 bytes, seguido pelo byte ModR/M, o byte SIB, o deslocamento de até 4 bytes e um valor imediato, também de 4 bytes. Todas os argumentos são opcionais, com exceção do opcode. O arranjo da instrução pode ser visto na figura 5[5].

Mesmo em modo 64-bits, sem um prefixo REX a maioria das instruções não pode acessar mais do que os 8 registradores da arquitetura x86-32, e não consegue acessar dados com

Figure 5: Formato de instrução na arquitetura x86-64.



largura maior que 32 bits. Em contrapartida, como mencionado anteriormente, sem o bit REX é possível acessar os bytes altos dos registradores AX a DX.

O prefixo REX tem o propósito de permitir não apenas que 16 registradores de uso geral sejam acessados, mas que seus operandos tenham 16 bits. Como dito anteriormente ele garante que todos os registradores possam ser acessados de maneira uniforme, a um pequeno custo de flexibilidade. Embora não é possível acessar os bytes altos dos registradores AX a DX, é possível acessar os bytes baixos de cada um dos registradores de uso geral, o que antes não era possível. A especificação de acesso de registradores menores que 64 bits é idêntico ao da arquitetura x86-32.

Na tabela 5.1 vemos os bits do prefixo REX, e na figura 6 vemos exemplos do uso do prefixo no endereçamento de instruções, retirados de [3]. Referimos a esse texto para mais detalhes.

5.2 Instruções de ponto flutuante

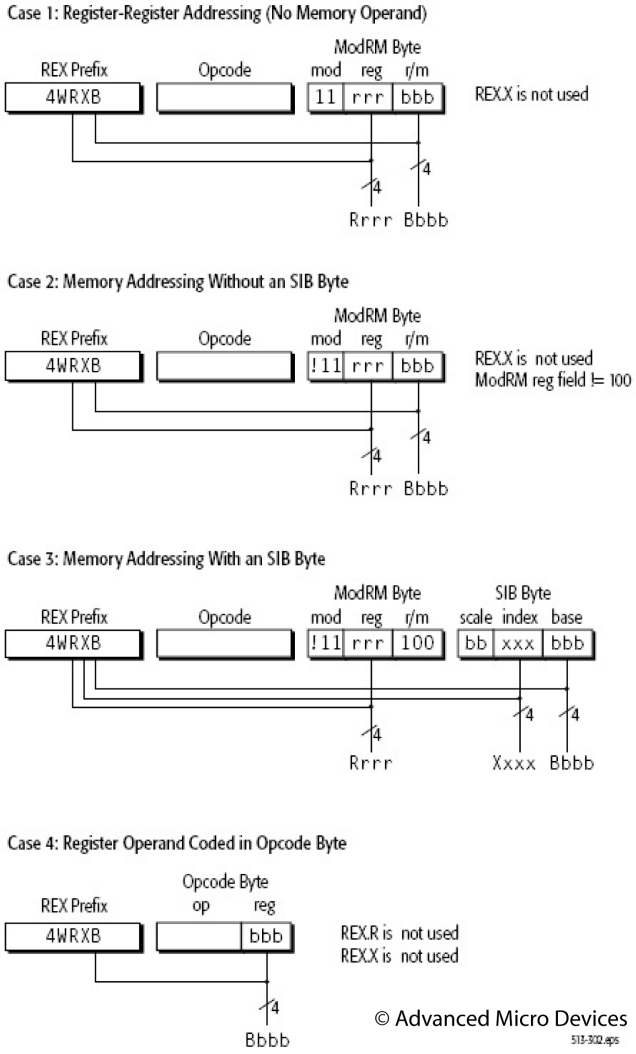
A arquitetura x86-64 oferece três maneiras de operar sobre dados de ponto flutuante. Uma pelas instruções SSE (Streaming SIMD[3] Extension), pelas instruções multimídia de 64 bits (3dNow!, MMX) e pelas instruções de ponto flutuante x87. Embora a funcionalidade seja replicada, a própria descrição da arquitetura recomenda que as instruções SSE

[3]Single Instruction Multiple Data

Mnemônico	Posição	Definição
-	7-4	0100
REX.W	3	0=Tamanho de operando padrão 1=Tamanho de operando de 64 bits
REX.R	2	1=Extensão do campo reg do *byte* ModR/M, permitindo acesso a 16 registradores
REX.X	1	1=Extensão do campo *index* do byte SIB, permitindo acesso a 16 registradores
REX.X	0	1=Extensão do campo *r/m* do byte ModR/M, permitindo acesso a 16 registradores

Table 2: Os bits do prefixo REX.

Figure 6: Exemplos do uso do prefixo REX em modo 64-bits.



sejam utilizadas no lugar das outras. Em particular, devido ao fato de que nas instruções SSE uma única instrução pode operar sobre até 4 números de ponto flutuante de precisão simples, e duas de precisão dupla. Outra razão para focar nas instruções SSE(2/3) é que na especificação da arquitetura foram acrescentadas 8 novos registradores XMM, que são utilizados nessas instruções.

A operação e endereçamento das funções de ponto flutuante é igual à da arquitetura x86-32, com a exceção de que para as instruções SSE é possível utilizar um registrador REX para endereçar os novos registradores XMM, de maneira análoga a que foi vista para os registradores de uso geral.

Existe algum consenso de que as instruções de ponto flutuante da arquitetura x86-32 necessitavam de consolidação, em particular o FPU x87. Este utiliza um mecanismo de pilha para executar instruções de ponto flutuante, e é considerado bastante arcaico em comparação a outros mecanismos de ponto flutuante. Como as instruções SSE tem uma vantagem de desempenho, alguns sistemas operacionais pra arquitetura x86-64 utilizam apenas os registradores XMM pra operações de ponto flutuante. A versão de 64 bits do Windows XP por exemplo não salva os registradores de ponto flutuante nem os registradores MMX durante mudanças de contexto.

6. O BIT NX

Bits que previnem a execução de dados existem na arquitetura Intel desde o 80286. O problema dessa implementação é que ela era amarrada na segmentação. Como sistemas operacionais modernos não utilizam a segmentação, preferindo ver o espaço de dados como sendo "achatado", e como essa visão é passada para os programas, programas modernos não podiam fazer proveito dessa tecnologia.

Para corrigir esse problema a AMD, na sua implementação da arquitetura x86-64 introduziu o **bit NX** (*No eXecute*), às vezes descrito como uma tecnologia "anti-vírus" pela empresa. A Intel, por sua vez, utiliza o mesmo mecanismo, mas com o nome de **bit XD** (*eXecute Disable*). Vamos referir a esse recurso como bit NX. Consiste de um bit na tabela de tradução de páginas que é carregado sempre que uma página é carregada. Se a página sendo acessada tem o bit NX igual a um, o processador acusa um General Protection Exception.

Essa alteração vai ter um efeito em particular contra *exploits* do tipo *buffer overflow*, embora não elimine esse problema completamente.

7. DIFERENÇAS ENTRE AMD E INTEL

No momento da escrita desse artigo os conjuntos de instruções dos processadores AMD64 e EMT64 são idênticos, apesar de que inicialmente haviam pequenas diferenças – algumas instruções implementadas em apenas um processador ou outro. A primeira implementação da Intel não tinha as operações LAHF e SAHF (Carrega/armazena bits de status no registrador AH), enquanto que introduzia uma instrução não implementada pela AMD, o CMPXCHG16B (compare e troque 16 bytes). Como nenhum dessas instruções é crucial a incompatibilidade era mínima. Contanto, apenas os processadores AMD64 mais recentes implementam instruções SSE3, que existiam em processadores Pentium há mais tempo, enquanto que os primeiros processadores que implementavam EMT64 não vinham com o bit NX.

8. CONCLUSÃO

As mudanças introduzidas na ISA x86-64 não foram muitas; mudanças no número de instruções e suas capacidades, uma pequena mudança no mecanismo de endereçamento e acesso a registradores e integração de instruções SSE como parte do *core* da arquitetura. Claro que associado a isso está o espaço de endereçamento de 64 bits, apresentado como um bloco contíguo pela arquitetura para facilitar o endereçamento pelo sistema operacional. Ainda são poucos os sistemas que operam nessa arquitetura, e são poucos softwares que utilizam os 64 bits. Apesar disso, o aumento de espaço de endereçamento foi bastante grande. Uma mudança mais drástica é a sugestão pela parte da AMD (e a implementação da Microsoft) de se deixar de utilizar o FPU x87 para processar valores de ponto flutuante, a fim de oferecer uma interface só para esse tipo de operação. Também interessante é a revigoração do mecanismo de proteção contra execução indesejada, que, embora já existente na arquitetura de 32 bits não era utilizável.

9. REFERENCES

[1] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 1: Applications Programming*, 2005.

[2] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, 2005.

[3] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 3: General Purpose And System Instructions*, 2005.

[4] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture*, 2005.

[5] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual Volume 2A: Instruction Set Reference, A-M*, 2005.