

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MANUELA KLANOVICZ FERREIRA

**Estudo e Modelagem de Instruções de
Virtualização Intel VT-x para Arquitetura
MIPS R3000**

Trabalho de Conclusão de Curso apresentado
como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Prof. Dr. Nicolas Bruno Maillard
Orientador

Prof. Dr. Philippe O. A. Navaux
Co-orientador

Porto Alegre, julho de 2008

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Ferreira, Manuela Klanovicz

Estudo e Modelagem de Instruções de Virtualização Intel VT-x para Arquitetura MIPS R3000 / Manuela Klanovicz Ferreira. – Porto Alegre: Instituto de Informática da UFRGS, 2008.

55 f.: il.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Rio Grande do Sul. Curso de Bacharelado em Ciência da Computação, Porto Alegre, BR-RS, 2008. Orientador: Nicolas Bruno Maillard; Co-orientador: Philippe O. A. Navaux.

1. Virtualização. 2. Linguagens de Descrição de Arquiteturas. 3. Simuladores. I. Maillard, Nicolas Bruno. II. Navaux, Philippe O. A..

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. José Carlos Ferraz Hennemann

Pró-Reitor de Coordenação Acadêmica: Prof. Pedro Cezar Dutra Fonseca

Diretor do Instituto de Informática: Prof. Flávio Rech Wagner

Chefe do Departamento de Informática Teórica: Prof. Leila Ribeiro

Chefe do Departamento de Informática Aplicada: Prof. José Valdeni de Lima

Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

*Essa felicidade que supomos,
Árvore milagrosa, que sonhamos
Toda arreada de dourados pomos,*

*Existe, sim: mas nós não a alcançamos
Porque está sempre apenas onde a pomos
E nunca a pomos onde nós estamos.*

— VELHO TEMA - OLAVO BILAC

AGRADECIMENTOS

Agradeço em primeiro lugar e em especial a minha mãe, Matilde Klanovicz, pelo apoio que ela sempre me deu e por aquele que ela ainda dará, pois sei que sempre vou poder contar com ela.

Agradeço também:

Ao meu irmão, Rodrigo Klanovicz Ferreira, por ter mantido por muito tempo o computador de casa operacional e disponível, realizando eventuais manutenções. Além disso agradeço o incentivo que recebi dele para começar a usar Linux.

Ao meu namorado, Mauricio Machado da Silva, que está comigo desde que eu comecei a estudar para passar no vestibular e sempre foi compreensivo quando eu passava dias somente estudando.

A todos os amigos que fiz aqui na UFRGS principalmente a Gigi, o KO, o Félix, o Santagada e a Bárbara, que entraram no curso no mesmo semestre que eu e sempre estiveram ao meu lado.

Ao meu Professor Orientador, Nicolas Maillard, por me orientar na elaboração deste trabalho, estando sempre disponível para eventuais dúvidas.

Ao Professor Navaux, meu Co-orientador nesse trabalho, e ao Henrique Freitas, por me oferecerem uma oportunidade de bolsa no grupo de Processamento Paralelo e Distribuído do Instituto de Informática da UFRGS, através da qual aprendi muito mais sobre arquiteturas paralelas e também tive a oportunidade de participar de diversos eventos nessa área e publicar artigos, tenho certeza que essa experiência fará uma grande diferença na minha vida profissional e acadêmica.

Obrigada!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS	9
LISTA DE ALGORITMOS	11
LISTA DE FIGURAS	13
LISTA DE TABELAS	15
RESUMO	17
1 INTRODUÇÃO	19
1.1 Importância da Virtualização	19
1.2 Necessidade do Estudo do Suporte de Hardware à Virtualização	20
1.3 Objetivos e Contribuições	21
1.4 Estrutura do Texto	21
2 CONTEXTO CIENTÍFICO	23
2.1 Conceitos de Virtualização	23
2.2 Técnicas de Virtualização	24
2.3 Virtualização de Hardware e de Software	25
2.3.1 Suporte de Software	26
2.3.2 Suporte de Hardware	26
2.4 Linguagens de Descrição de Arquiteturas	26
2.5 Simuladores	27
3 FERRAMENTAS UTILIZADAS	29
3.1 SystemC	29
3.2 ArchC	29
3.2.1 Descrição do Elemento AC_ARCH	30
3.2.2 Descrição do Elemento AC_ISA	31
3.2.3 Ferramentas do ArchC	32
4 TECNOLOGIA DE VIRTUALIZAÇÃO DA INTEL	35
4.1 Estruturas de Controle	35
4.2 Instruções de Virtualização	37
4.3 Considerações sobre a Intel VT-x	38
5 MIPS R3000 COM INSTRUÇÕES DE VIRTUALIZAÇÃO	41
5.1 Modelo Original do MIPS	41
5.2 Modelo do MIPS-vt	42

6	SIMULAÇÃO UTILIZANDO MIPS-VT	45
6.1	Descrição do Programa de Teste	45
6.2	Execução e Resultados do Programa Teste	47
7	CONCLUSÃO	49
	REFERÊNCIAS	51
	ANEXO DOCUMENTAÇÕES	55

LISTA DE ABREVIATURAS E SIGLAS

ADL	Architecture Description Language
Ap	Aplicação
HDL	Linguagem de Descrição de Hardware
ns	Nano Segundo
MMV	Monitor de Máquina Virtual
MV	Máquina Virtual
RTL	Register Transfer Level
SO	Sistema Operacional
ULA	Unidade Lógica e Aritimética
VMCS	Virtual Machine Control Structure
VMX	Virtual Machine Extensions

LISTA DE ALGORITMOS

3.1	Exemplo de elemento AC_ARCH sem pipeline de instruções.	31
3.2	Exemplo de arquivo de descrição do formato de instruções.	31
3.3	Exemplo de descrição de comportamento da instrução ADD com pipeline de cinco estágios.	32
5.1	Em C++, <code>struct</code> que modela o VMCS	42
6.1	Código resumido do programa teste.	46
6.2	Código resumido da descrição de comportamento da instrução VMXON, com exemplos de <i>traces</i>	48
6.3	Saída do <i>trace</i> acrescentado à descrição de comportamento da instrução VMXON.	48

LISTA DE FIGURAS

Figura 1.1:	Exemplos de vantagens de virtualização (UHLIG et al,2005). Aplicação (Ap) e Hardware (HW).	20
Figura 2.1:	Relacionamento entre MMV e MVs, com MMV diretamente sobre o hardware.	23
Figura 2.2:	Virtualização Total (à esquerda) com SOs convidados em suas versões originais, acreditando que estão executando diretamente sobre o hardware. Paravirtualização (à direita) com SOs modificados que estão cientes de serem virtualizados. Aplicação (Ap)	24
Figura 2.3:	Arquitetura de MMV hospedada.	25
Figura 2.4:	Classificação das ADLs segundo a natureza da informação que fornecem.	27
Figura 2.5:	Classificação de Simuladores.	28
Figura 3.1:	Geração de ferramentas no ArchC.	33
Figura 4.1:	Modo VMX <i>root</i> e VMX <i>non-root</i> (UHLIG et al,2005).	35
Figura 4.2:	MV entrada e MV saída.	36
Figura 4.3:	Representação de uma MV saída e a emulação das instruções feita pelo MMV.	36
Figura 4.4:	Transições provocadas pelas instruções de operação da Intel VT-x. . .	38
Figura 5.1:	Formato dos tipos de instruções do MIPS R3000.	41
Figura 5.2:	Exemplo de estado da CPU no MIPS-vt.	43
Figura 6.1:	Diagrama de execução do teste.	45

LISTA DE TABELAS

Tabela 4.1:	Instruções para a manipulação do VMCS.	37
Tabela 4.2:	Instruções de operação do VMCS.	38
Tabela 5.1:	Registradores específicos do MIPS-vt.	42
Tabela 5.2:	Instruções de virtualização do MIPS-vt.	44
Tabela 6.1:	Estatísticas da execução total do teste.	47
Tabela 6.2:	Resumo das estatísticas referentes à execução de cada instrução. . . .	47

RESUMO

Os simuladores estão entre as ferramentas mais utilizadas, tanto por pesquisadores como por estudantes, para facilitar o entendimento de uma nova tecnologia. Entender o funcionamento das novas tecnologias de hardware, como o suporte de hardware à virtualização, é necessário para garantir a sua correta utilização. Com a simulação da virtualização de hardware é possível alcançar esse entendimento e também detectar possíveis deficiências e desenvolver novas soluções. A virtualização está sendo cada vez mais utilizada, isso torna importante a necessidade do seu estudo em busca de seu melhor desempenho, principalmente da virtualização de hardware, pois outros estudos mostram que a virtualização de software possui resultados de desempenho superiores à virtualização de hardware. A capacidade que a virtualização permite de executar múltiplos sistemas operacionais em uma mesma plataforma física tem como um de seus benefícios eliminar a necessidade da compra de equipamentos diferentes para cada sistema operacional requisitado, além de aumentar o aproveitamento de equipamentos, que, por estarem executando mais de uma máquina virtual, ficam menos tempo ociosos.

O objetivo deste trabalho é estudar a tecnologia de virtualização da Intel para arquiteturas IA-32, a Intel VT-x, para permitir a sua modelagem adaptada sobre um simulador de processador MIPS, criando um novo simulador com suporte de hardware à virtualização. Com isso, espera-se suprir a falta de simuladores para arquiteturas de suporte à virtualização.

Este trabalho descreve um simulador de processador MIPS que fornece suporte à virtualização de hardware, o MIPS-vt. Esse suporte foi inspirado na tecnologia Intel VT-x. O MIPS-vt foi feito utilizando a ferramenta ArchC, versão 2.0, e é uma extensão de um modelo do MIPS R3000. O ArchC 2.0 é uma linguagem de descrição de arquitetura capaz de gerar simuladores.

Palavras-chave: Virtualização, Linguagens de Descrição de Arquiteturas, Simuladores.

1 INTRODUÇÃO

Com a rapidez na evolução das tecnologias de hardware, torna-se difícil para programadores e, até mesmo, pesquisadores entenderem o seu funcionamento. Para auxiliar nesse entendimento, são utilizadas ferramentas de descrição de arquiteturas e simuladores. Entre o lançamento de uma nova arquitetura e o desenvolvimento de ferramentas e modelos que auxiliem no seu entendimento, há um longo percurso. Entender o funcionamento das novas tecnologias de hardware é necessário para garantir a sua correta utilização. É através do estudo das novas arquiteturas que pode-se detectar suas deficiências e desenvolver novas soluções.

1.1 Importância da Virtualização

Virtualização pode ser vista como a capacidade de executar múltiplos sistemas operacionais (SOs) em uma única plataforma física, dividindo os recursos de hardware. Em outras palavras, a virtualização permite ter duas ou mais máquinas virtuais (MVs), executando SOs diferentes simultaneamente em um mesmo computador de forma totalmente isolada.

Os benefícios clássicos da virtualização incluem maior aproveitamento, gerenciamento e confiabilidade de *mainframes*. Vários usuários que necessitem de diferentes SOs podem facilmente compartilhar um servidor virtualizado e falhas em softwares convidados podem ser isoladas nas MVs em que ocorreram (UHLIG et al, 2005). Na Figura 1.1, são ilustrados três exemplos de vantagens da virtualização.

O isolamento dos softwares em suas próprias MVs oferecido pela virtualização pode aumentar a segurança e confiabilidade de diversos sistemas. A segurança pode ser aumentada com o confinamento de invasões nas MVs em que ocorrem, impedindo que ela atinja outras aplicações que estejam executando em outras MVs. Enquanto a confiabilidade pode ser melhorada porque falhas de software em uma VM não afetam as outras MVs. Além disso, para assegurar maior tolerância a falhas é possível executar cópias idênticas da mesma carga de trabalho em duas MVs distintas e, assim, em caso de falha de uma das MVs, a outra pode fazer a cobertura instantaneamente.

A consolidação de diversos SOs em um mesmo servidor elimina a necessidade de *data centers* possuírem um servidor para cada SO, até mesmo para cada versão do mesmo SO solicitada. Com a virtualização é possível executar os SOs necessários em uma mesma plataforma física, permitindo uma redução do custo e do desperdício dos recursos de hardware, através da alocação dinâmica de recursos para a MV que mais necessitar deles naquele momento.

Se, por exemplo, houver um servidor de e-mail, que normalmente é mais utilizado

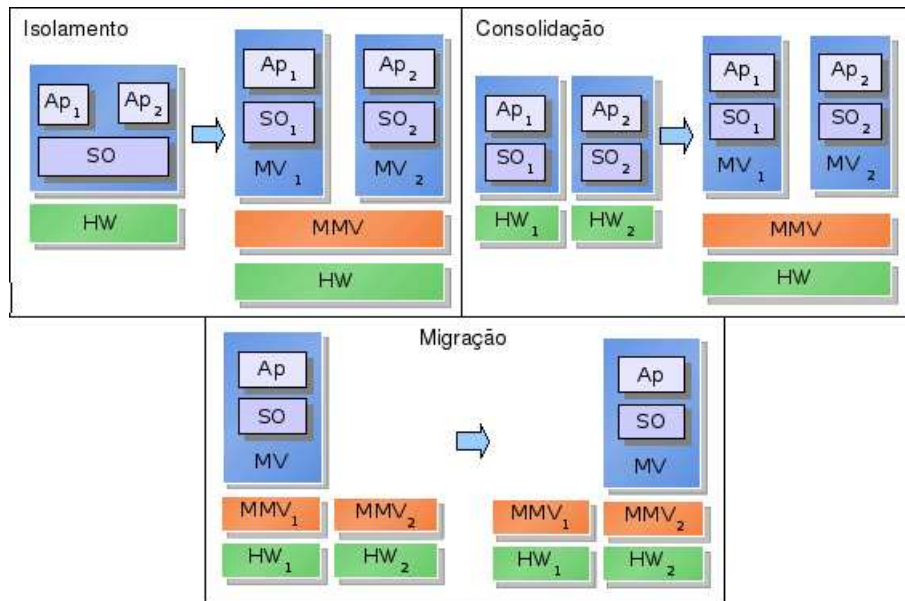


Figura 1.1: Exemplos de vantagens de virtualização (UHLIG et al,2005). Aplicação (Ap) e Hardware (HW).

durante o dia, executando em uma plataforma Linux na MV1 e um programa de *backup* de arquivos, normalmente executado à noite, em uma plataforma Windows na MV2, os dois executariam na mesma plataforma física ao invés de haver um servidor para cada aplicação com seus respectivos SOs. Isso evitaria a necessidade de se comprar um segundo servidor, além de evitar o desperdício, durante a noite, dos recursos de um servidor que apenas abrigasse um serviço de e-mail e, durante o dia, de um servidor que abrigasse apenas um serviço de *backup* que é executado somente à noite.

A consolidação também apresenta a vantagem de desenvolvedores de software podem criar ambientes virtuais para rodar múltiplos SOs, ou mesmo diferentes versões do mesmo SO para testar o software. E como outra vantagem pode-se citar o uso em ambientes com pouco espaço em disco, como notebooks, onde, para economizar esse espaço, os usuários que precisam utilizar mais de um SO instalam o menos utilizado como uma MV.

Através do encapsulamento do estado do SO convidado feito pelas MVs, a virtualização torna possível fazer a migração do estado de SOs convidados do hardware onde está atualmente para uma plataforma diferente. Assim, todo o estado de um SO convidado pode ser salvo e carregado em outra MV localizada na mesma ou em outra plataforma física, isso facilita muito a manutenção e atualização dos sistemas.

Assim, com o crescente uso e os variados benefícios fornecidos pela virtualização, cresce também a necessidade de seu estudo para garantir uma implementação simples e correta.

1.2 Necessidade do Estudo do Suporte de Hardware à Virtualização

A virtualização é um conceito bem definido desde a década de 60, quando a IBM desenvolveu a primeira MV com o objetivo de aumentar o aproveitamento dos recursos através do seu compartilhamento (ROSENBLUM,GARFINKEL,2005). Tendo sido restrita a servidores de grande porte, a virtualização tornou-se comum em computadores

peçoais. Isso ocorreu devido ao aumento da capacidade de processamento desses computadores e também devido a técnicas de software e, mais recentemente, de hardware que foram desenvolvidas e empregadas para aumentar o desempenho dos sistemas virtualizados. Por ser uma tecnologia recente, ainda não existem ferramentas que auxiliem no estudo ou simulação do suporte de hardware à virtualização.

Os sistemas virtualizados utilizando somente técnicas de hardware apresentam desempenho inferior aos sistemas virtualizados utilizando apenas técnicas de software. Isso surpreende, pois era de se esperar que as técnicas de hardware obtivessem melhores resultados. Uma das causas desse fato pode ser a utilização incorreta do suporte de hardware, outra possibilidade descrita em (ADAMS, AGESSEN, 2006) seria porque o suporte de hardware oferecido pelos processadores para a virtualização não é utilizado em conjunto com as já existentes e bem estabelecidas técnicas de software.

Isso evidencia a necessidade de novos estudos relacionados ao suporte de hardware existentes atualmente para a virtualização. Entretanto, apesar da necessidade de pesquisas relacionadas a essas novas arquiteturas com suporte à virtualização, não foram encontradas ambientes de simulação que possibilitem o estudo destas arquiteturas.

1.3 Objetivos e Contribuições

Sob o contexto apresentado anteriormente um dos objetivos deste trabalho é estudar a tecnologia de virtualização da Intel para arquiteturas IA-32, a Intel VT-x. E, através deste estudo, modelar a Intel VT-x de forma adaptada sobre um simulador de MIPS, criando um simulador com suporte de hardware à virtualização. Para a modelagem foi utilizada a ferramenta ArchC 2.0. Um modelo MIPS R3000 foi estendido com as principais características da tecnologia Intel VT-x. Esse novo modelo, denominado MIPS-vt, foi utilizado na simulação e análise de um teste para verificar sua validade e o seu correto funcionamento.

Com o MIPS-vt, espera-se suprir a falta de ambientes de simulação para estudo da virtualização em hardware. É possível simular teste escritos em assembler para verificar a melhor maneira de utilização desta arquitetura com suporte de hardware à virtualização. Além disso, pesquisadores podem alterar o modelo MIPS-vt para identificar problemas de projeto ou novas instruções para suporte à necessidades ainda não descritas.

1.4 Estrutura do Texto

A fim de contextualizar o leitor, o Capítulo 2 trata das principais técnicas de virtualização de software e de hardware. Neste Capítulo, são também discutidas algumas das Linguagens de Descrição de Arquiteturas e Simuladores. No Capítulo 3, são apresentadas as ferramentas utilizadas neste trabalho: ArchC e SystemC. As estruturas de controle e instruções da tecnologia Intel VT-x são descritas no Capítulo 4.

Com o Capítulo 5, inicia a descrição do MIPS-vt, com a descrição do MIPS R3000 original, seguida da descrição do modelo MIPS-vt. Então há a descrição da simulação utilizando o MIPS-vt no Capítulo 6. E finalmente, no Capítulo 7, as conclusões deste trabalho.

No anexo documentações, são relacionadas as publicações feitas ao longo do desenvolvimento deste trabalho e a página utilizada para relatar o andamento do trabalho.

2 CONTEXTO CIENTÍFICO

Como contexto científico deste trabalho, serão apresentadas as técnicas de virtualização de software e de hardware empregadas para possibilitar a virtualização de SOs não modificados executando em arquiteturas x86. Posteriormente são discutidas algumas Linguagens de Descrição de Arquiteturas e como elas contribuem para a modelagem de sistemas. Em relação a simuladores de processadores é feita uma discussão sobre os atualmente disponíveis, suas vantagens e limitações.

2.1 Conceitos de Virtualização

Como foi dito, virtualização é a capacidade de executar múltiplos SOs em uma mesma plataforma física ao mesmo tempo, dividindo os recursos de hardware. Na Figura 2.1, pode-se ver o relacionamento entre alguns conceitos de virtualização definidos abaixo.

- Máquina Virtual (MV) é uma abstração da máquina física real oferecida. É uma porção dos recursos de hardware totalmente isolada das demais MVs do sistema, permitindo que o mesmo computador seja compartilhado como se fosse vários (ROSE,2004), mas cada um com uma porção dos recursos reais.
- Monitor de Máquinas Virtuais (MMV) ou hipervisor é o software que gerencia a distribuição dos recursos de hardware para cada SO convidado, criando um ambiente virtual isolado (máquina virtual) para cada um.

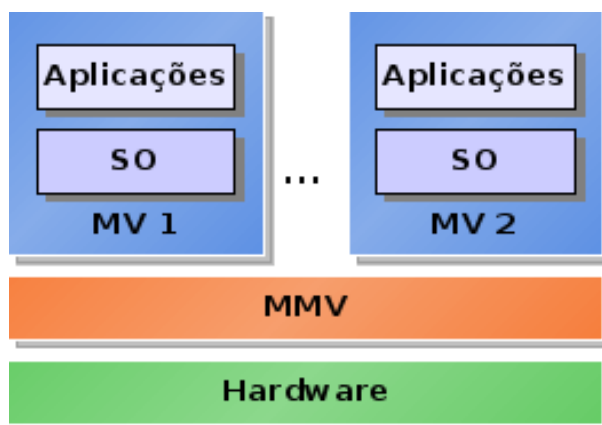


Figura 2.1: Relacionamento entre MMV e MVs, com MMV diretamente sobre o hardware.

2.2 Técnicas de Virtualização

Existem dois critérios para classificar a virtualização de sistemas. O primeiro leva em consideração o fato de os SOs convidados serem modificados para a virtualização ou não. Segundo este critério, existem os seguintes tipos de virtualização (ROSE,2004):

1. Virtualização total (*full virtualization*) é aquela que permite virtualizar SOs não modificados, pois replica virtualmente toda a arquitetura do hardware.
2. Paravirtualização é a virtualização em que o SO convidado é modificado para poder executar concorrentemente com outros SOs que também foram modificados para a virtualização.

A Figura 2.2 ilustra os tipos de virtualização quando levado em consideração o fato de o SO convidado ser modificado ou não.

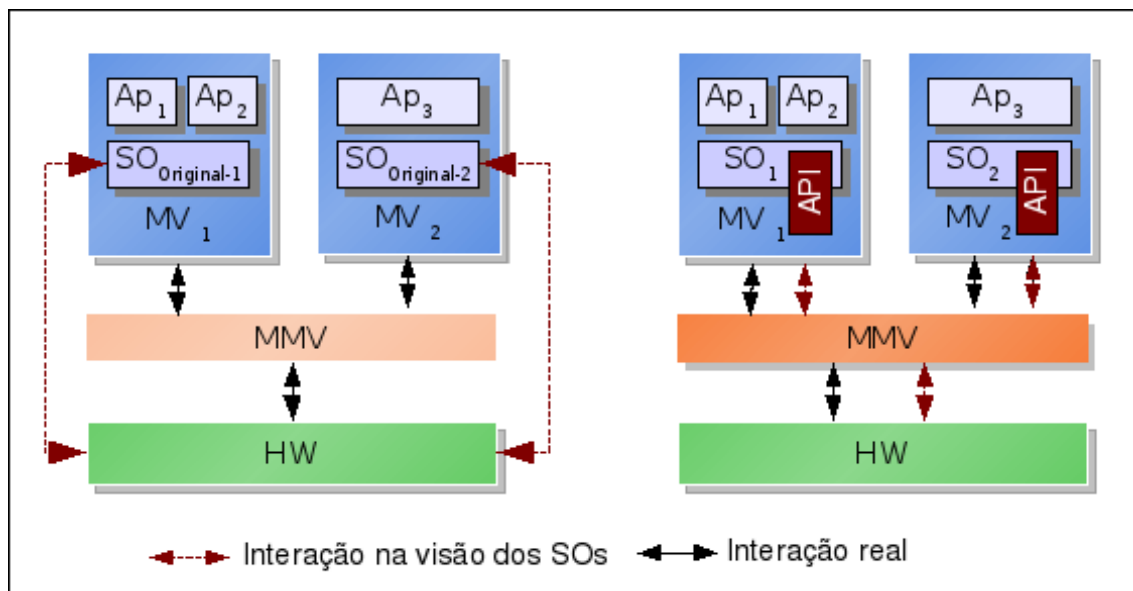


Figura 2.2: Virtualização Total (à esquerda) com SOs convidados em suas versões originais, acreditando que estão executando diretamente sobre o hardware. Paravirtualização (à direita) com SOs modificados que estão cientes de serem virtualizados. Aplicação (Ap)

A paravirtualização sempre demonstrou desempenho superior ao da virtualização total, pois os SOs modificados, ao invés de tentarem solicitar diretamente a CPU a execução de instruções sensíveis, solicitavam a execução dessas instruções ao MMV, evitando o teste para verificar se cada instrução era ou não sensível. Instruções sensíveis são aquelas que não devem ser executadas pelas MVs, pois podem afetar o estado de outras MVs.

Entretanto, com a utilização do atual suporte de hardware fornecido à virtualização, a virtualização total e a paravirtualização têm apresentado desempenhos equivalentes. O VMWare ESX 3.0.1, um MMV que utiliza virtualização total, e o XenEnterprise 3.2, que utiliza paravirtualização, apresentam desempenhos semelhantes ao executar *benchmarks* como SPECcpu2000, Passmark e Netperf, que juntos simulam as principais cargas de trabalho de um *datacenter* (XEN SOURCE,2007). Ambos os MMVs citados aproveitam o suporte de hardware à virtualização fornecido pelos processadores atuais.

O segundo critério de classificação para a virtualização de sistemas considera onde o MMV está sendo executado, diretamente no hardware ou sobre um SO hospedeiro.

1. Arquitetura de Máquina Virtual Hospedada (*Hosted Virtual Machine Architecture*) ocorre quando o MMV executa sobre um SO hospedeiro, como pode-se ver na Figura 2.3. O VMWare é um MMV que executa sobre um SO hospedeiro para aproveitar o suporte que o hospedeiro oferece a grande variedade de dispositivos disponíveis atualmente.
2. Arquitetura de Máquina Virtual Não Hospedada que é quando o MMV executa diretamente sobre o hardware, como é possível ver na Figura 2.1.

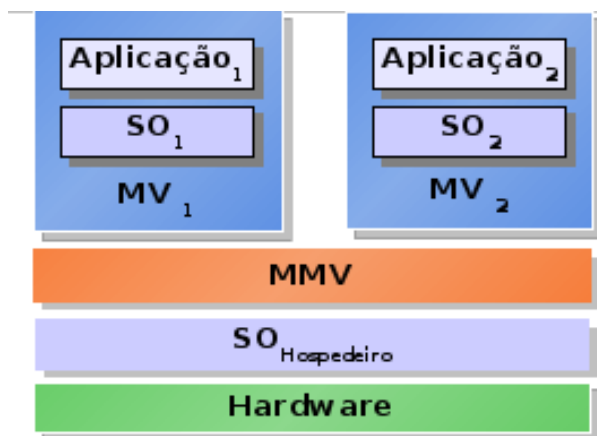


Figura 2.3: Arquitetura de MMV hospedada.

2.3 Virtualização de Hardware e de Software

Em (POPEK,GOLDBERG,1974), os autores afirmaram que "*Um MMV pode ser construído para qualquer computador se o conjunto de instruções sensíveis deste computador for um subconjunto das instruções privilegiadas deste computador*", onde instruções sensíveis são instruções que em um contexto de virtualização podem interferir na execução de outros SOs que compartilham os recursos de hardware, comprometendo o isolamento entre os SOs convidados (ROSE,2004). Essas instruções devem ser detectadas pelo MMV que deve emulá-las de maneira a não comprometer o isolamento. Popek e Goldberg também definiram três características que eles acreditavam serem essenciais para arquiteturas de máquinas virtualizáveis:

1. Qualquer programa executando sobre um MMV deve ter os mesmos efeitos demonstrados se o programa fosse executado diretamente na máquina original, exceto em relação ao tempo de execução e à disponibilidade de recursos.
2. Um subconjunto fixo grande das instruções deve ser executado diretamente pelo processador real. Essas instruções devem ser as não sensíveis.
3. O MMV deve estar no controle completo dos recursos do sistema.

Essas características se mostraram difíceis de serem alcançadas em determinadas arquiteturas, como a arquitetura x86, da qual a arquitetura IA-32 faz parte.

A arquitetura x86 apresenta alguns obstáculos à virtualização, pois não obedece às premissas de Popek e Goldberg. Os principais desafios para a virtualização desta arquitetura são (ADAMS,AGESEN,2006):

- *Visibilidade do estado de privilégio.* Quem deve executar no nível de maior privilégio é o MMV. Nas arquiteturas x86, o SO convidado pode observar que ele está desprivilegiado quando ele lê o seu código seletor de segmento (%cs), pois o nível de privilégio corrente está armazenado nos dois bits menos significativos do %cs.
- *Falta de interrupções de software (traps) quando instruções privilegiadas executam em nível de usuário.* Por exemplo, o código privilegiado `popf` ("pop flags") pode mudar os flags da Unidade Lógica e Aritmética (ULA) e do sistema (e.g. IF, que controla a entrega de interrupções). Para SOs convidados rodando em modo desprivilegiado precisa-se que a instrução `popf` provoque uma interrupção de software, assim o MMV pode emular o seu efeito em um IF virtual. Infelizmente, um `popf` apenas não modifica o IF, nenhuma interrupção de software ocorre.

Nesta seção serão resumidamente apresentadas as principais técnicas de software e modificações de hardware utilizadas para contornar esses desafios da virtualização de arquiteturas x86.

2.3.1 Suporte de Software

Os obstáculos semânticos para a virtualização do x86 podem ser superados se o SO convidado executar sobre um intérprete ao invés de diretamente na CPU física. O intérprete pode prevenir a visibilidade do estado de privilégio referenciando um nível de privilégio virtual ao invés do estado de privilégio real na interpretação de instruções que não causam interrupção de software como `popf`. Executar sobre um intérprete, entretanto, pode causar um grande *overhead* que diminui bastante o desempenho.

A principal solução de software para isso é a tradução binária (*binary translation*) que faz a tradução das instruções necessárias em tempo de execução a partir do código binário, combinando precisão de interpretação e agilidade. Quando uma instrução necessita ser modificada, a tradução binária faz um desvio para outro trecho de código que execute essa instrução de forma controlada, e depois retorna para a execução normal das instruções. Tudo isso é feito em tempo de execução.

2.3.2 Suporte de Hardware

Recentemente, mudanças de arquitetura permitiram uma outra forma de solucionar os obstáculos da virtualização em arquiteturas x86. Entre as fabricantes que lançaram processadores com suporte de hardware à virtualização estão: a Intel, com suas arquiteturas VT-x (32 bits) (INTEL CORP VT-x,2005) e VT-i (64 bits) (INTEL CORP VT-i,2005); e a AMD, com os processadores pertencentes à arquitetura AMD Pacífica (64 bits) (AMD,2005). Ambas são muito semelhantes, seguindo os mesmos princípios.

Esse suporte de hardware à virtualização se concentra na troca de contexto entre o MMV e os SOs convidados e na detecção de instruções executadas pelos SOs convidados que devem ser interceptadas pelo MMV, pois são instruções sensíveis.

2.4 Linguagens de Descrição de Arquiteturas

As *Architecture Description Languages* (ADLs) são usadas para especificar arquiteturas programáveis. A especificação pode ser usada na simulação da arquitetura em questão e os resultados dessa simulação usados para modificar a especificação com o objetivo de encontrar a melhor arquitetura para um conjunto de aplicações. As ADLs também são

usadas para gerar protótipos de hardware especificando características como a velocidade do *clock* por exemplo.

Uma das características mais importante das ADLs está no fato de poder-se especificar uma arquitetura e então simulá-la, melhorá-la e validá-la antes de construir o hardware propriamente dito, economizando tempo e diminuindo os custos para a criação de arquiteturas. Também pode-se citar a possibilidade usar módulos já desenvolvidos por outras pessoas ou empresas, fazendo com que o tempo gasto na especificação das arquiteturas diminua.

Linguagens de Descrição de Hardware (HDLs) não costumam ser utilizadas para a descrição de arquiteturas pois não possuem abstração suficiente para descrevê-las no seu nível de sistema. Pode-se entender a estrutura da arquitetura através de HDLs, contudo, fica muito difícil entender o seu conjunto de instruções.

As ADLs podem ser classificadas segundo a natureza de informação que fornecem. Assim, pode-se classificar as ADLs como estruturais, comportamentais ou mistas (MISHRA,DUTT,2005). Veja Figura 2.4.

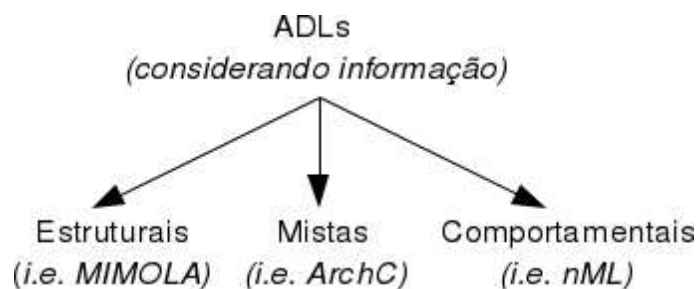


Figura 2.4: Classificação das ADLs segundo a natureza da informação que fornecem.

As ADLs estruturais descrevem a estrutura em termos de arquitetura de componentes e sua conectividade, como exemplo de ADL estrutural, pode-se citar a MILOLA (BASHFORD et al,1994), criada na Universidade de Dortmund (Alemanha). Uma vantagem da MIMOLA é que uma mesma descrição pode ser usada para simulação, geração de teste e compilação.

Já as ADLs comportamentais descrevem o comportamento do conjunto de instruções da arquitetura do processador e ignoram estruturas de hardware detalhadas. Nessa classe há a nML (FAUTH et al,1995), originária da Technical University of Berlin (Alemanha), cujos projetistas identificaram o fato de que algumas instruções compartilham propriedades e utilizam um sistema de hierarquia para descrever o conjunto de instruções.

Quando fala-se das ADLs mistas, refere-se a linguagens que se preocupam com os dois aspectos apresentados anteriormente, ou seja, se preocupam com a estrutura dos componentes e com o comportamento do conjunto de instruções, esse é o caso do ArchC (ARCHC TEAM,2004), utilizado neste trabalho.

2.5 Simuladores

A utilização de simuladores para facilitar a compreensão do funcionamento de processadores ou modelos de arquiteturas tem sido constante recentemente. Isso porque os simuladores facilitam o entendimento da manipulação de estruturas de um sistema. É possível observar o funcionamento real de um sistema real, imitado pelo simulador, e verificar como as instruções e/ou estruturas são manipuladas.

As três principais características de um simulador são: o escopo que ele abrange e o nível de detalhamento que proporciona (WOLFFE et al,2002). Cada uma dessas características origina duas classes de simuladores, como pode ser visto na Figura 2.5.

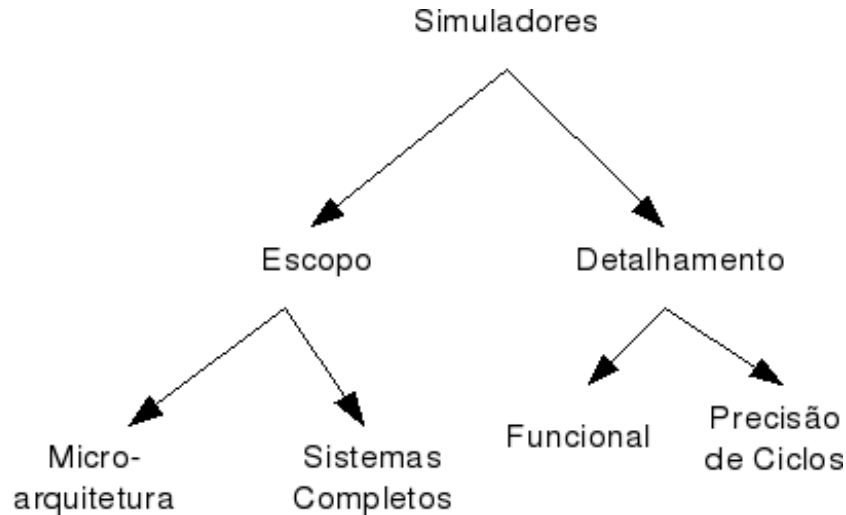


Figura 2.5: Classificação de Simuladores.

Com relação ao escopo, os simuladores podem ser simuladores de microarquiteturas, que simulam apenas um microprocessador; ou simuladores de sistemas completos (*full-system simulators*), capazes de simular sistemas completos levando em consideração não só o processador, como o sistema de memória e os dispositivos de entrada e saída.

Referente ao nível de detalhamento, os simuladores podem ser funcionais, que não possuem precisão de ciclo e apenas têm interesse que as instruções tenham resultados corretos; ou podem ser com precisão de ciclo e informações de clock, normalmente com pipeline, interessados que as instruções executem no tempo correto.

Existem inúmeros simuladores para arquiteturas superescalares, entre eles pode-se citar o SimpleScalar (SIMPLESCALAR,2008), que permite simulação funcional e com precisão de ciclos e é um simulador de sistemas completos. O SimpleScalar é um pacote de simuladores para a análise de arquiteturas superescalares, com um conjunto de oito simuladores para a realização de diferentes análises, como nas previsões de desvio, *cache* e execução especulativa.

Como foi dito anteriormente, a modelagem de processadores feita por ADLs pode ser usada para a simulação. É o caso do ArchC que permite a modelagem de uma determinada arquitetura para sua posterior simulação. Os simuladores gerados pelo ArchC podem ser funcionais ou com precisão de ciclos. Com relação ao escopo, os simuladores gerados pelo ArchC podem ser tanto de micro-arquiteturas como de sistemas completos, nesse caso levando em consideração hierarquias de memória e dispositivos de entrada e saída.

3 FERRAMENTAS UTILIZADAS

Neste capítulo são apresentadas as duas ferramentas utilizadas no desenvolvimento deste trabalho. Primeiramente são descritas as principais características e evolução do ArchC e do SystemC. Posteriormente são descritas as principais ferramentas do ArchC.

3.1 SystemC

O SystemC é uma extensão da linguagem C++ que visa elevar o nível de abstração para projeto e verificação de hardware(RIGO et al,2005). O SystemC é totalmente baseado em C/C++ e o código para a biblioteca de simulação está disponível em domínio público na Internet (SYSTEMC,2008). SystemC é composto de um conjunto de classes C++, que estendem a linguagem para permitir a modelagem de hardware e sistemas. Possibilita a modelagem em níveis de abstração mais baixos, como RTL (*Register Transfer Level*), abstração que descreve operações em circuitos digitais e é utilizada em linguagens de descrição de hardware como VHDL (ASHENDEN,1990). O principal objetivo do SystemC, entretanto, não é substituir as linguagens de descrição de hardware (i.e. VHDL), mas sim, possibilitar o projeto em nível de sistema.

Embora SystemC suporte vários modelos de computação, comunicação e níveis de abstração, não é possível extrair de uma descrição genérica em SystemC de um processador toda a informação necessária para gerar automaticamente ferramentas de software, que permitam ao projetista experimentar e avaliar um novo conjunto de instruções.

A versão mais antiga do SystemC, a 1.0, possuía como estruturas de descrição portas e módulos; e como mecanismos de comunicação e sincronização apenas sinais de hardware. A versão 2.0 tem como principal melhoria novos mecanismos de comunicação e sincronização como canais, interfaces e eventos (SYSTEMC TEAM,2002). A versão do SystemC utilizada neste trabalho foi a versão 2.2, que é praticamente idêntica à versão 2.0, apenas com alguns problemas corrigidos.

3.2 ArchC

O ArchC (ARCHC TEAM,2004) é uma linguagem de descrição de arquiteturas que usa tanto informações estruturais quanto de conjunto de instrução para a geração automática de simuladores, sendo que é a primeira a gerar simuladores em alto nível de abstração descritos em SystemC. Seu principal objetivo é prover informações suficientes, em um nível de abstração adequado, para permitir aos usuários explorar e verificar uma nova arquitetura através da geração automática de ferramentas de software como montadores, simuladores, interfaces de depuração e comunicação.

O ArchC foi escolhido por ser uma ADL que se preocupa tanto com aspectos estruturais como comportamentais das arquiteturas modeladas, possibilitando uma modelagem mais completa dos sistemas.

Basicamente, uma descrição de arquitetura em ArchC é dividida em duas partes: a descrição do conjunto de instruções (AC_ISA) e a descrição dos recursos de arquitetura (AC_ARCH). Na descrição AC_ISA, o projetista fornece os detalhes sobre formatos, tamanhos e nomes de instruções combinados com toda a informação necessária para a decodificação e o comportamento das instruções. A descrição AC_ARCH informa ao ArchC quais são os dispositivos de armazenamento, estrutura de *pipeline* e alguns outros detalhes da estrutura da arquitetura. Baseando-se nessas duas descrições, cria-se ferramentas capazes de gerar simuladores interpretados, escritos em SystemC, simuladores compilados, escritos em C++, e montadores redirecionados para a arquitetura descrita. Entre as arquiteturas modeladas utilizando o ArchC destacam-se as arquiteturas MIPS, Sparc-V8 e o Intel 8051, um microcontrolador com arquitetura CISC e instruções multi-ciclo de tamanho variável, muito utilizado em sistemas embarcados, todas disponíveis em (ARCHC,2008).

No início deste trabalho, foi utilizada a versão 1.6 da ferramenta ArchC. Com o lançamento da versão 2.0, que tem como diferencial ser capaz de simular (utilizando o SystemC) um processador multi-core, optou-se por utilizar a versão 2.0. Ambas as versões estão disponíveis em (ARCHC,2008). Em (ARAÚJO et al,2005), é descrito como implementar processadores com múltiplos núcleos utilizando ArchC 2.0.

A seguir serão esclarecidos alguns detalhes sobre como são feitas as declarações dos elementos AC_ARCH e AC_ISA.

3.2.1 Descrição do Elemento AC_ARCH

ArchC necessita de algumas informações estruturais sobre os recursos disponíveis na arquitetura a fim de gerar automaticamente ferramentas de software. O projetista deve fornecer tais informações na descrição do AC_ARCH.

O nível do detalhe usado para esta descrição dependerá do nível de abstração desejado para o modelo. Por exemplo, um projetista pode querer simular um conjunto de instruções da arquitetura MIPS, mas sem considerar o pipeline, como pode ser visto no Algoritmo 3.1. De fato, isto é exatamente como os projetistas iniciam um modelo novo da arquitetura em ArchC. Os modelos sem sincronismo, ou precisão de ciclo, e sem a informação de pipeline são chamados de funcionais. Tal modelo simula o comportamento de um conjunto de instruções, executando todas as operações de uma instrução dada durante um único ciclo de *clock*. Construindo um modelo funcional, os projetistas recolhem bastante conhecimento sobre o ISA para descrever um modelo mais detalhado, evitando propagar os erros que poderiam ter sido reparados previamente na fase de projeto. A descrição da arquitetura em nível funcional exige descrições de instruções de comportamento muito simples, porém demanda poucas informações estruturais.

Algoritmo 3.1: Exemplo de elemento AC_ARCH sem pipeline de instruções.

```

1 AC_ARCH(r3000){
2
3     ac_wordsize 32;
4     ac_mem      DM:5M;
5
6     ac_regbank  RB:34;
7
8     ARCH_CTOR(r3000) {
9         ac_isa("r3000_isa.ac");
10        set_endian("big");
11
12    };
13 };

```

Para os modelos com precisão de ciclo e descrição de pipeline, são adicionadas no elemento AC_ARCH informações referentes a quantidade e nomeação de cada estágio de pipeline.

3.2.2 Descrição do Elemento AC_ISA

A descrição do Conjunto de Instruções do ArchC insere todas as informações necessárias para se sintetizar um decodificador, junto com algumas considerações referentes ao comportamento de cada instrução. Essa descrição é dividida em dois arquivos diferentes, um contendo as declarações dos formatos das instruções e outro contendo as informações referentes ao comportamento delas.

As instruções modeladas podem ser agrupadas em tipos. O agrupamento das instruções leva em consideração formatos e operações comuns executadas por certas instruções, como por exemplo o adiantamento de dados no pipeline (*forwarding*). No arquivo onde são descritos os formatos das instruções também são descritos os formatos dos tipos das instruções. No Algoritmo 3.2 pode-se ver a declaração do tipo de instrução `Type_R` na linha 2. Na linha 4, a instrução `ADD` é declarada como sendo do tipo `Type_R`. A declaração do formato a instrução `ADD` pode ser vista na linha 9, sendo que o código da operação é definido em hexadecimal na linha 10.

Algoritmo 3.2: Exemplo de arquivo de descrição do formato de instruções.

```

1 AC_ISA(r3000){
2     ac_format Type_R = "%op:6 %rs:5 %rt:5 %rd:5 %shamt:5 %func:6";
3
4     ac_instr<Type_R> add;
5
6     ISA_CTOR(r3000){
7
8         /* outras declaracoes */
9         add.set_asm("add %reg, %reg, %reg", rs, rt, rd);
10        add.set_decoder(op=0x00, func=0x20);
11
12    };
13 };

```

No arquivo onde estão descritos os comportamentos das instruções são aceitas duas funções de comportamento que são totalmente independentes da arquitetura do projeto (RIGO et al, SBAC-2004). São elas:

- `ac_behavior(begin){}`

- `ac_behavior(end) {}`

O `ac_behavior(begin)` é executado antes do início da simulação. Ele pode ser usado para inicializar os registradores do processador ou endereços de memória, por exemplo. O `ac_behavior(end)` é executado após extremidades da simulação. Ambos os `ac_behavior(begin)` e `ac_behavior(end)` têm acesso a todos os recursos declarados da arquitetura, assim como quaisquer outros comportamentos.

Cada tipo de instrução declarado pode ter uma descrição de comportamento. Essa descrição será executada em instruções do tipo em questão antes da descrição de comportamento específica da instrução.

Caso o modelo de arquitetura possua precisão de ciclo e pipeline é necessário acrescentar a cada descrição de comportamento de instrução um `switch` com uma opção de execução para cada estágio do pipeline. No Algoritmo 3.3, é mostrada a descrição de comportamento para a instrução `ADD` em um pipeline de cinco estágios.

Algoritmo 3.3: Exemplo de descrição de comportamento da instrução `ADD` com pipeline de cinco estágios.

```

1 void ac_behavior( add ){
2     switch( stage ) {
3     case IF:
4         break;
5
6     case ID:
7         break;
8
9     case EX:
10        EX_MEM.alures = ex_value1 + ex_value2;
11
12        // Teste de overflow
13        if ( ((ex_value1 & 0x80000000) == (EX_MEM.alures & 0x80000000)) &&
14            ((EX_MEM.alures & 0x80000000) != (ex_value2 & 0x80000000)) ) {
15            fprintf(stderr, "EXCEPTION(add): integer overflow.\n");
16            exit(EXIT_FAILURE);
17        }
18
19        EX_MEM.regwrite = ID_EX.regwrite;
20        EX_MEM.memread = ID_EX.memread;
21        EX_MEM.memwrite = ID_EX.memwrite;
22        EX_MEM.rdest = ID_EX.rd;
23        break;
24
25    case MEM:
26        break;
27
28    case WB:
29        break;
30
31    default:
32        break;
33    }

```

3.2.3 Ferramentas do ArchC

O ArchC possui um conjunto de ferramentas que, a partir de uma descrição de um processador, geram automaticamente montadores e simuladores para esse modelo. Essas

ferramentas são:

- `acasm` gera o montador para uma arquitetura previamente descrita.
- `acsim` gera o simulador para uma arquitetura previamente descrita.

Como pode ser visto na Figura 3.1, com os fontes da descrição de uma arquitetura, o `acasm` gera um montador e o `acsim` gera o simulador para essa arquitetura. Com o assembler de um programa teste como entrada, o montador é capaz de gerar uma descrição de código que pode ser chamada de binário. Com o binário do teste como entrada, o simulador é capaz de gerar estatísticas referentes à simulação do teste. Essas estatísticas possuem informações sobre o tempo de execução, o número total de instruções executadas e o número individual de cada instrução executada.

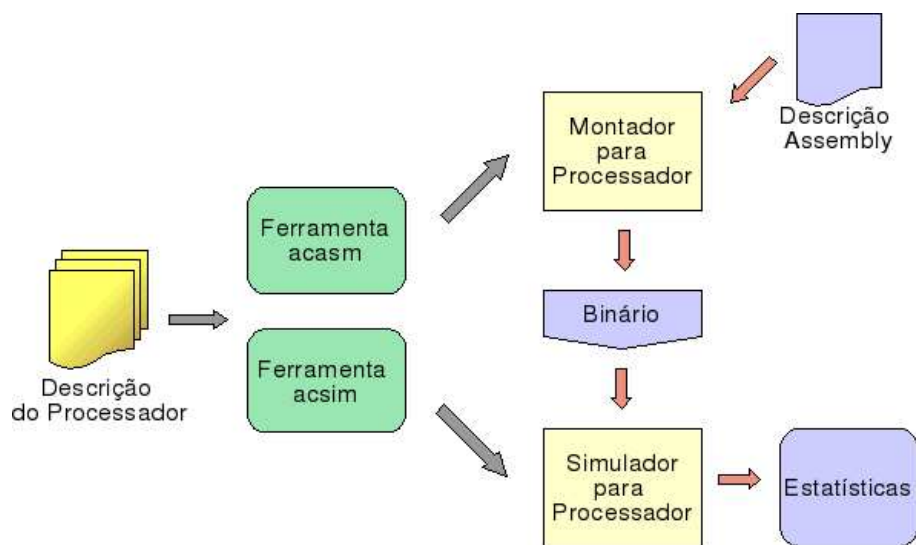


Figura 3.1: Geração de ferramentas no ArchC.

4 TECNOLOGIA DE VIRTUALIZAÇÃO DA INTEL

A tecnologia Intel VT-x consiste em um conjunto de instruções e registradores, denominado *Virtual Machine Extensions* (VMX), para suporte à virtualização em arquiteturas IA-32. Esta seção apresenta quais estruturas de controle e instruções fazem parte desta tecnologia.

4.1 Estruturas de Controle

VMX inclui a oferta de dois novos modos de operação da CPU chamados *VMX root*, onde o MMV executa, e *VMX non-root*, onde os SOs convidados executam. Ambos suportam todos os quatro níveis de privilégio (UHLIG et al,2005), como pode-se ver na Figura 4.1, permitindo que os SOs convidados executem em um nível zero de privilégio e acreditem que possuem o controle da CPU. Essa capacidade simplifica o desenvolvimento dos MMV. Com isso, também é possível que o MMV execute em múltiplos estágios de privilégio.

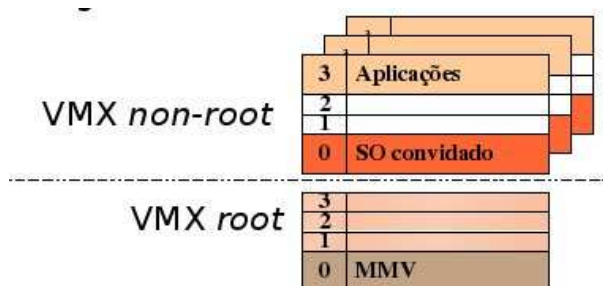


Figura 4.1: Modo VMX root e VMX non-root (UHLIG et al,2005).

Esta tecnologia define duas novas transições: MV entrada, que é a transição do modo VMX root para o modo VMX non-root, e MV saída que faz a transição inversa, veja Figura 4.2. Também é disponibilizada uma nova estrutura de dados, o *Virtual Machine Control Structure* (VMCS). Cada MV gerada possui o seu VMCS para guardar o seu contexto, sendo que apenas um deles será o VMCS ativo. Com a configuração do VMCS é possível definir quais instruções irão causar MV saídas e também é no VMCS que ficam salvos os motivos que ocasionaram uma MV saída.

VMCS ativo é aquele pertencente à MV que está sendo virtualizada naquele momento. A CPU pode estar executando instruções do MMV ou da MV cujo contexto está salvo no VMCS ativo. Se uma VM está sendo executada na CPU é porque o VMCS que guarda o seu contexto é o VMCS ativo. Assim, sempre que se desejar modificar a MV que está sendo executada, é necessário modificar o VMCS ativo.

O VMCS é dividido em duas seções, a área de estado do convidado e a área de estado do hospedeiro. Cada MV entrada irá salvar o estado da CPU na área de estado do hospedeiro no VMCS e carregar o novo estado da área de estados do convidado no VMCS. A MV saída irá salvar o estado do processador na área de estado do convidado no VMCS e carregar o estado da área de estado do hospedeiro no VMCS.



Figura 4.2: MV entrada e MV saída.

O comportamento do processador no modo *VMX root* é semelhante ao comportamento de quando as extensões VMX estão desabilitadas. As principais diferenças são que as instruções VMX estão disponíveis e que os valores que podem ser carregados em determinados registradores de controle são limitados.

O comportamento do processador no modo *VMX non-root* é restrito e foi modificado para facilitar a virtualização. Ao invés de sua operação normal, certas instruções e eventos causam VMX saídas do MMV. Por causa dessas VM saídas, a funcionalidade de um software no modo *VMX non-root* são limitadas. Algumas instruções não podem ser executadas no modo *VMX non-root* porque causam incondicionalmente MV saídas, essas instruções incluem: `CPUID` e `MOV from CR3`. Esta limitação permite ao MMV um determinado controle sobre os recursos do sistema. Utilizando os campos do VMCS, é possível configurar outras instruções, interrupções e exceções para causar MV saídas de forma condicional.

Os SOs convidados não têm como descobrir que estão executando em modo *VMX non-root*, pois o bit com essa informação não pode ser acessado por eles. Caso isso aconteça, a instrução de consulta é interceptada pelo MMV e é provocada uma MV saída para passar o controle da CPU ao MMV (INTEL CORP VT-x, 2005). Quando o MMV toma o controle da CPU, sabendo o motivo pelo qual ocorreu uma MV saída, o MMV pode simular a instrução que ocasionou essa MV saída no contexto do SO convidado, salvo no VMCS indicado pelo registrador de VMCS ativo.

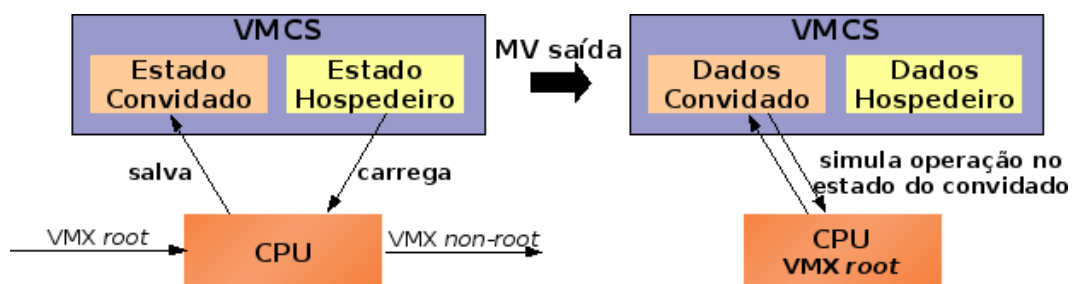


Figura 4.3: Representação de uma MV saída e a emulação das instruções feita pelo MMV.

Na Figura 4.3, é representada uma MV saída. O estado da CPU passa de *VMX non-root* para *VMX root*, enquanto o estado do SO convidado é salvo na área de estado do convidado do VMCS e o estado do MMV é carregado da área de estado do hospedeiro do VMCS. A direita da Figura 4.3, vê-se que enquanto o MMV executa no modo *VMX root*

ele altera o estado do SO convidado, simulando a execução da instrução que ocasionou a MV saída.

4.2 Instruções de Virtualização

VMX inclui cinco novas instruções para manipular o VMCS e cinco instruções que são operações VMX. Esta seção mostra quais são essas instruções, seus formatos e descreve o seus comportamentos. As novas instruções que realizam a manipulação do VMCS (INTEL CORP VT-x,2005) oferecidas pela tecnologia Intel VT-x podem ser vistas na Tabela 4.1:

Tabela 4.1: Instruções para a manipulação do VMCS.

Nome	Formato	Descrição
VMPTRLD	<i>vmptrld addr</i>	Possui apenas um operando fonte que está na memória, esse operando é um ponteiro para uma instância de um VMCS. Esta instrução torna o ponteiro para um VMCS passado o ponteiro para o VMCS ativo.
VMPTRST	<i>vmptrst addr</i>	Possui apenas um operando destino localizado na memória. Ela salva o ponteiro de VMCS ativo no destino passado como parâmetro.
VMCLEAN	<i>vmclean addr</i>	Possui um único operando localizado em memória. Torna o VMCS passado como parâmetro inativo.
VMREAD	<i>vmread reg/addr addr</i>	Lê um campo do VMCS (o código deste campo é passado em um registrador), cuja referência foi passada no segundo operando, e salva no destino que pode ser um registrador ou na memória, cuja referencia foi passada no primeiro parâmetro.
VMWRITE	<i>vmwrite addr reg/addr</i>	Escreve em um campo do VMCS (o código deste campo é passado em um registrador), cuja referência é passada no primeiro operando. O valor do operando fonte que pode ser um registrador ou uma posição de memória e sua referência é passada como segundo operando.

As instruções de operação estão descritas na Tabela 4.2. As instruções *VMCALL*, *VMLAUNCH* e *VMRESUME* não possuem operandos pois suas ações sempre afetam o VMCS referenciado como ativo. Na Figura 4.4, estão ilustradas as mudanças de estado provocadas pelas instruções de operação da tecnologia Intel VT-x. Pode-se ver que há três modos de operação: operação normal, onde as instruções de virtualização não são reconhecidas, exceto a instrução *VMXON*; o modo *VMX root* e o modo *VMX non-root*.

Uma outra característica importante da tecnologia Intel VT-x é a capacidade de detectar instruções sensíveis. Em (UHLIG et al,2005) é dito que esse controle sobre instruções sensíveis é gerenciado pelo VMCS, mas não dá detalhes de como isso é feito. Há algumas instruções sensíveis que sempre causam MV saídas, mas na tecnologia Intel VT-x

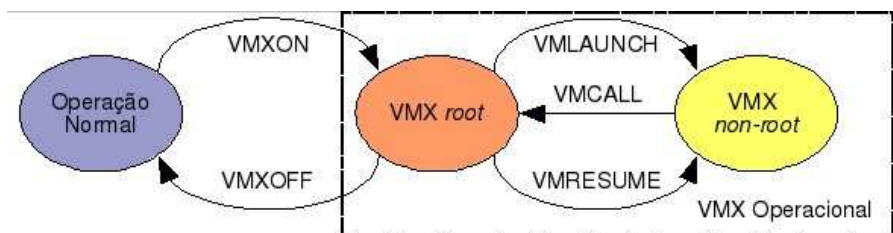


Figura 4.4: Transições provocadas pelas instruções de operação da Intel VT-x.

Tabela 4.2: Instruções de operação do VMCS.

Nome	Formato	Descrição
VMCALL	vmcall	Permite que o SO convidado no modo VMX <i>non-root</i> solicite algum serviço do MMV. Uma transição MV saída ocorre, passando o controle para o MMV no modo VMX <i>root</i> . Esta é a única instrução de virtualização que pode ser executada em modo VMX <i>non-root</i> sem causar erro.
VMLAUNCH	vmlaunch	Executada no modo VMX <i>root</i> , lança uma MV gerenciada pelo VMCS ativo. Ocorre uma MV entrada, passando para o modo VMX <i>non-root</i> e o controle é passado para a MV lançada.
VMRESUME	vmresume	Executada no modo VMX <i>root</i> , retoma à execução da MV gerenciada pelo VMCS ativo. Ocorre uma MV entrada, passando para o modo VMX <i>non-root</i> e o controle é passado para a MV apontada pelo VMCS ativo.
VMXON	vmxon	Habilita as instruções de virtualização do processador, passando do modo e operação normal, sem instruções de virtualização, para o modo VMX <i>root</i> .
VMXOFF	vmxoff	Desabilita as instruções de virtualização do processador, passando do modo de operação VMX <i>root</i> para o modo de operação normal.

também é possível configurar outras instruções para causarem MV saídas, bastando para isso alterar alguns campos do VMCS.

4.3 Considerações sobre a Intel VT-x

A tecnologia Intel VT-x fornece suporte à virtualização através de instruções e estruturas que facilitam a troca de contexto entre as MVs e o MMV. Através dessas instruções, é possível salvar o contexto das MVs, nas transições de MV saída, com uma instrução apenas, da mesma maneira como é possível salvar o contexto do MMV nas MV entradas.

Outro ponto importante, é a detecção, por hardware, da execução de instruções sensíveis. A Intel VT-x intercepta por padrão diversas instruções sensíveis como a `CPUID`, e permite, através da configuração dos campos do VMCS, que outras instruções sejam adicionadas a esse conjunto de instruções sensíveis, provocando uma VM saída caso sejam executadas em modo VMX *non-root*. Com a MV saída, o MMV passa a ter o controle da

CPU e pode realizar as operações necessárias para que a instrução sensível seja executada de forma controlada. Com a interceptação das instruções sensíveis feita por hardware, não é mais necessário que o MMV teste cada instrução executada pelas MVs para conferir se ela é sensível.

A Intel VT-x não é uma tecnologia trivial. Assim seria interessante a existência de ferramentas que facilitassem o seu entendimento. Mesmo porque, como já foi dito anteriormente, apesar que a Intel VT-x ser implementada em hardware, ela perde em desempenho para técnicas de software, e isso pode ser devido a utilização incorreta das técnicas de hardware.

5 MIPS R3000 COM INSTRUÇÕES DE VIRTUALIZAÇÃO

O MIPS R3000 foi escolhido como o modelo de processador a ser modificado porque, como uma arquitetura RISC, possui instruções simples e regulares, com operandos que estão localizados em registradores ou são valores imediatos pequenos. A seção a seguir apresenta o modelo original do MIPS, feito em ArchC 2.0. A seção 5.2 mostra como a tecnologia Intel VT-x foi modelada como uma extensão da arquitetura MIPS R3000, descrevendo como é o modelo MIPS-vt.

5.1 Modelo Original do MIPS

O modelo original, em ArchC versão 2.0, criado para simulação do processador MIPS R3000 está disponível em (ARCHC,2008). É basicamente o modelo que foi descrito em (HENNESSY et al,1998). Originalmente, o modelo possuía cinco estágios de pipeline: busca (IF), decodificação (ID), execução (EX), escrita na memória (MEN) e escrita no banco de registradores (WB).

O modelo implementa os 32 registradores de 32 bits para uso geral. Também possui os registradores `hi` e `lo`, que juntos somam 64 bits, para operações de divisão e multiplicação.

Nesse modelo do MIPS, não foram implementados os registradores de coprocessamento, como o registrador de nível de privilégio de CPU. Como esses registradores não estão presentes, as instruções para acesso a esses registradores também não estão presentes, como é o caso da instrução de `MOV`. O modelo possui todas as instruções aritméticas e de desvio do MIPS implementadas. A instrução `SYSCALL` e `BREAK` estão modeladas, mas não estão implementadas, provocando a parada da simulação se executadas.

Há três tipos de instruções, que possuem 32 bits cada. Na Figura 5.1 são ilustrados os formatos e tamanhos dos campos de cada tipo de instrução.

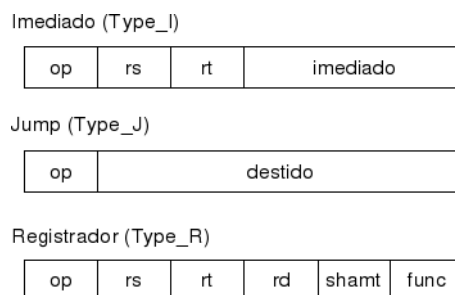


Figura 5.1: Formato dos tipos de instruções do MIPS R3000.

5.2 Modelo do MIPS-vt

Esta seção descreve como o modelo do processador MIPS R3000 foi estendido para oferecer suporte à virtualização. Esse modelo criado com instruções de virtualização é denominado MIPS-vt. Nenhuma das estruturas ou instruções do MIPS R3000, descritas na Seção 5.1, foi removida, logo o modelo continua possuindo pipeline e cinco estágios. A tecnologia Intel VT-x, descrita no Capítulo 4 deste trabalho, foi adaptada para ser implementada sobre o modelo de processador do MIPS R3000 descrito anteriormente.

O VMCS foi implementado como uma estrutura específica do processador, sendo uma `struct` em C++. O Algoritmo 5.1 mostra a `struct` que implementa o VMCS. Na linha 6, a lista nomeada como `VM_exit_cond_instruction` é a lista de instruções consideradas sensíveis. Para que as instruções sejam consideradas sensíveis e não possam ser executadas pelas MVs, é necessário que acrescentar seus nomes à lista e recompilar o modelo.

Ao invés de modelar o VMCS com duas áreas de estado, uma para o SO convidado e outra MMV, o MIPS-vt salva o contexto do MMV em um VMCS especial, referenciado por um novo registrador específico, e o contexto das MVs são salvos nos demais VMCS. O modelo possui capacidade de operar com até dez MV, pois esse é o número de VMCS disponibilizados.

Algoritmo 5.1: Em C++, `struct` que modela o VMCS

```

1 struct VMCS
2 {
3     //campos de controle
4     bool active;
5     int launch_state;
6     set<string> VM_exit_cond_instruction;
7
8     //estrutura para salvar o contexto
9     int rb[RB_MAX_REG];
10    int pc;
11
12    VMCS();
13    void clear();
14    void print();
15 };

```

Foram criados quatro novos registradores de 32 bits. Esses registradores foram declarados no elemento `AC_ARCH` descrito na Seção 3.2.1, logo abaixo do banco de registradores original do MIPS R3000. A Tabela 5.1 descreve esses registradores.

Tabela 5.1: Registradores específicos do MIPS-vt.

Nome	Descrição
RS_VMX_OP	Indica se as instruções de virtualização estão ativas (ON) ou inativas (OFF).
RS_VMX_MODAL	Indica o modo de operação do processador quando as instruções VMX estão ativas. Pode ser VMX <i>root</i> e VMX <i>non-root</i> .
RS_VMCS_ATIVO	Referencia o VMCS ativo.
RS_VMCS_HOST	Referencia o VMCS que guarda o contexto do MMV.

Apenas um subconjunto das instruções VMX foi modelado, as instruções VMREAD e VMWRITE não foram modeladas. Essas instruções não foram modeladas pois suas funções eram, respectivamente, ler e escrever campos do VMCS. Entretanto, no MIPS-vt, o VMCS está simplificado e possui apenas um campo que deve ser escrito. Esse campo indica o endereço do início do código da MV cujo contexto está salvo neste VMCS. Como não houve a necessidade de ler esse campo do VMCS, a instrução VMREAD não foi implementada. E a necessidade de escrever esse campo foi suprida através da alteração da instrução VMLAUNCH, que no modelo MIPS-vt possui um operando que é um endereço de memória para indicar o início do código da MV que está sendo lançada.

Na Figura 5.2 é possível ver um possível estado da CPU do MIPS-vt. No estado ilustrado, há três MV lançadas, cada uma com o seu VMCS. O VMCS ativo é o VMCS 2, pois é ele quem está sendo referenciado pelo registrador de VMCS ativo. O VMCS do MMV é aquele referenciado pelo registrador VMCS do MMV. Além do registrador de VMCS ativo e do registrador do VMCS do MMV, há os outros dois registradores que também foram adicionados à arquitetura do MIPS R3000. O registrador de Modo de Operação, registra se a CPU está executando em modo VMX *root* ou modo VMX *non-root*. No estado ilustrado na Figura 5.2, o modo de operação é VMX *root*, logo, é o MMV que está executando na CPU. O registrador de Operações VMX indica se as instruções VMX estão ativas (ON) ou inativas (OFF).

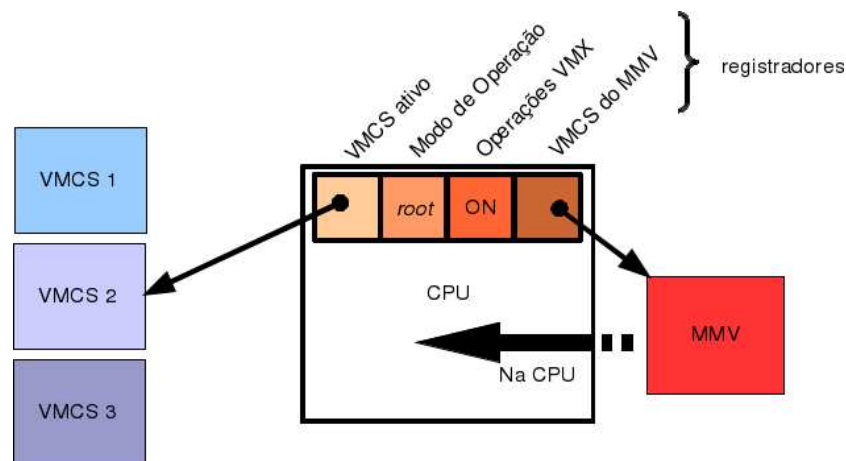


Figura 5.2: Exemplo de estado da CPU no MIPS-vt.

As instruções de virtualização modeladas são de tipos já existentes no modelo original do MIPS R3000, descritos na Seção 5.1. Na Tabela 5.2 está descrito o formato de cada instrução de virtualização implementada no MIPS-vt e o seu tipo. Essas instruções foram acrescentadas no elemento AC_ISA, no arquivo onde são descritos os formatos das instruções, descrito na Seção 3.2.2.

As instruções que possuem operandos foram implementadas como sendo do tipo Imediato. As instruções sem operando foram implementadas como sendo do tipo Registrador. Os tipos pré-existentes no modelo MIPS R3000 foram utilizadas para simplificar o modelo MIPS-vt, diminuindo o número de alteração necessárias para permitir virtualização em hardware.

Como foi explicado na Seção 4.2, a tecnologia Intel VT-x permite configurar quais instruções causam MV saídas, através de alterações dos campos do VMCS. Então, para que o MIPS-vt também fornecesse essa capacidade, foi adicionada como campo dos VMCSs uma lista de quais instruções causam MV saídas. Por questão de simplificação, não foram

Tabela 5.2: Instruções de virtualização do MIPS-vt.

Nome	Formato	Tipo
VMXON	vmxon	Registrador
VMXOFF	vmxoff	Registrador
VMCALL	vmcall	Registrador
VMRESUME	vmresume	Registrador
VMCLEAR	vmclear <i>imm(reg)</i>	Imediato
VMLAUNCH	vmlaunch <i>addr</i>	Imediato
VMPTRLD	vmptrld <i>imm(reg)</i>	Imediato
VMPTRST	vmptrst <i>imm(reg)</i>	Imediato

implementadas as instruções `VMREAD` e `VMWRITE` para escrever e ler campos do VMCS. Assim, caso queira-se acrescentar ou retirar instruções da lista de instruções sensíveis, é necessário fazer isso no código do VMCS e recompilar o modelo.

Para conseguir detectar se uma instrução foi configurada para causar MV saídas, foi adicionado um teste no início da descrição de comportamento de cada instrução. Caso a instrução esteja configurada para causar MV saída, a execução da MV que tenta executar esta instrução é interrompida e é realizada uma MV saída, passando o controle da CPU para o MMV no modo VMX *root*.

6 SIMULAÇÃO UTILIZANDO MIPS-VT

Nesta seção será descrito um exemplo de simulação utilizando o MIPS-vt e as estatísticas geradas pela sua execução. Inicialmente há a descrição do programa teste. Em seguida é descrito o ambiente de execução do teste juntamente com os resultados obtidos.

O teste executado tem como objetivo validar o modelo MIPS-vt, para garantir a sua correta execução.

6.1 Descrição do Programa de Teste

Em (CASAZZA et al,2006) é feita a afirmação de que os *benchmarks* atuais não são ideais para realizar um teste em sistemas virtualizados. Isso porque ao testar sistemas virtualizados é importante levar em consideração o número de máquinas virtuais e qual propriedade específica do sistema virtualizado se deseja testar. Por esse motivo, para testar o modelo de processador MIPS-vt foi utilizado um teste elaborado especificamente para esse sistema virtualizado.

O programa teste elaborado para o modelo MIPS-vt tem como objetivo verificar se as trocas de contexto entre as MVs e o MMV estão ocorrendo de forma correta.

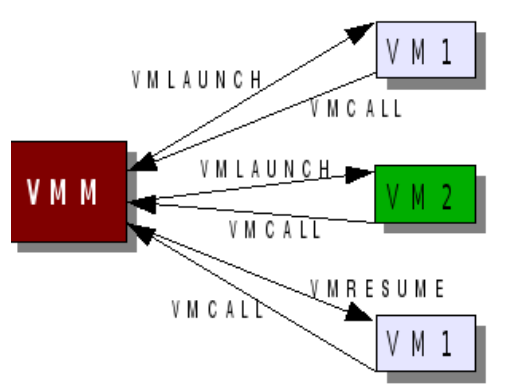


Figura 6.1: Diagrama de execução do teste.

Inicialmente, no programa teste, o MMV lança, utilizando a instrução **VMLAUNCH**, uma MV denominada MV1. Durante sua execução, a MV1 chama o MMV utilizando a instrução **VMCALL**, como ilustra a Figura 6.1. O MMV então lança a MV2. Depois do término da execução da MV2, o MMV é chamado utilizando a instrução **VMCALL**. O MMV então executa a instrução **VMRESUME** para retornar a execução da VM1, afim de que ela possa terminar.

Um resumo do código teste deste exemplo é apresentado no Algoritmo 6.1. O código completo está disponível na página no projeto (www.codeplex.com/visa).

Algoritmo 6.1: Código resumido do programa teste.

```

1 $main:
2   #habilita as operacoes VMX (ON)
3   vmxon
4
5   #lanca a MV1
6   addi $6, $0, 1
7   sw $6, 0($6)
8   #marca o VMCS da MV1 como VMCS ativo
9   vmptrld 0($6)
10  vmlaunch $mv1
11
12  # ... lanca MV2
13
14  #retoma execucao da MV1
15  addi $6, $0, 3
16  #marca VMCS da MV2 como VMCS ativo
17  vmptrld 0($6)
18  vmresume
19
20  #desabilita operacoes VMX (OFF)
21  vmxoff
22  break
23
24  #codigo fonte da MV1 calcula fatorial
25  $mv1:
26    addi $4, $0, 4    #$4=n=4
27    addi $5, $0, 1    #$5=f=1
28    vmcall            #return to main
29  $L1:
30    mult $5, $4        #$hi+$lo=f*n
31    mflo $5            #f=$lo
32    addi $4, $4, -1    #n = n-1
33    bne $4, $0, $L1    #if (n!=0) goto
34    vmcall
35  #codigo fonte da MV2

```

Primeiramente, no Algoritmo 6.1 é executada a instrução **VMXON**, linha 3, para habilitar as instruções de virtualização. Nas linhas 5 e 6, são feitas algumas inicializações para o lançamento da MV1. Na linha 9, é executado um **VMPTRLD** para tornar o VMCS da MV1 o VMCS ativo. Então a MV1 é lançada, na linha 10, com a instrução **VMLAUNCH**.

O código para lançamento da MV2 foi suprimido por ser semelhante ao código de lançamento da MV1. As únicas modificações necessárias são que o endereço passado como operando à instrução **VMPTRLD** deve ser o endereço do VMCS pertencente à MV2. E na instrução **VMLAUNCH**, o endereço passado como operando deve ser referente ao início do código fonte da MV2.

Na linha 16, o VMCS da MV1 é marcado como o VMCS ativo novamente e, na linha 18, o MMV faz um **VMRESUME** para retornar à execução da MV1 do ponto onde ela havia parado.

Na linha 21, as instruções de virtualização são desabilitadas através da instrução **VMXOFF**.

Das linhas 24 a 34 está o código fonte da MV1.

6.2 Execução e Resultados do Programa Teste

A execução do teste descrito na Seção 6.1 sobre o modelo MIPS-vt descrito na Seção 5.2, gerou um conjunto de estatísticas. Esse conjunto de estatísticas é gerado automaticamente pela ferramenta ArchC. As estatísticas possuem dois conjuntos de informações, um referente a dados da execução total do programa, e outro referente a dados da execução de cada uma das instruções.

Na Tabelas 6.2 e 6.1 estão resumidos o conjunto de estatísticas total da execução do programa teste e o conjunto de estatísticas referentes a cada instrução, respectivamente.

Tabela 6.1: Estatísticas da execução total do teste.

Nome	Quantidade
Tempo total da simulação	944 ns
Total de instruções executadas	53

Tabela 6.2: Resumo das estatísticas referentes à execução de cada instrução.

Instrução	Número total de execuções	Porcentagem do número de execuções
ADDI	15	28,3%
MULT	8	15,09%
MTLO	8	15,09%
SW	2	3,77%
BNE	8	15,09%
BREAK	1	1,89%
VMXON	1	1,89%
VMXOFF	1	1,89%
VMCALL	3	5,66%
VMLAUNCH	2	3,77%
VMRESUME	1	1,89%
VMPTRLD	3	5,66%
VMPTRST	0	0%

É possível verificar, observando as estatísticas, que a execução ocorreu conforme o previsto, logo o modelo está realizando as trocas de contexto de maneira correta. O número reduzido de instruções de virtualização deve-se ao fato de a troca de contexto ser feita utilizando apenas uma instrução quando o suporte à virtualização está presente. Quando se deseja trocar o contexto entre o MMV e uma MV é necessário apenas um VMLAUNCH ou VMRESUME que a troca de contexto, ou seja, o salvamento de todos os registradores e do contador de programa do MMV e o carregamento dos mesmos referentes à execução da MV é feito em apenas uma instrução.

Outros resultados importantes gerados pelo MIPS-vt são os *traces* gerados pela execução dos testes. Os *traces* podem ser adicionados facilmente à descrição de comportamento das instruções utilizando instrução `printf` da biblioteca `stdio` da linguagem de programação C. Assim, os usuários do modelo podem adicionar novos *traces* ou modificar os já existentes para que mostrem dados de acordo com suas necessidades. O Algoritmo

6.2 mostra um trecho da descrição de comportamento da instrução **VMXON**. Na linha 5, foi acrescentado um *trace* que informa o estado de execução do processador, *VMX root* ou *VMX non-root*, antes da execução da instrução **VMXON**.

Algoritmo 6.2: Código resumido da descrição de comportamento da instrução **VMXON**, com exemplos de *traces*.

```

1 void ac_behavior( vmxon ){
2     switch( stage ) {
3         /*descricao de comportamento de outros estagios*/
4         case EX:
5             printf( "RS_VMX_OP = %s\n", RS.read(RS_VMX_OP)? "ON": "OFF" );
6             // testa se esta operando em modo VMXON
7             if ( !RS.read(RS_VMX_OP) ) {
8                 /* restante da descricao de comportamento */
9             }
10            break;
11            /*descricao de comportamento de outros estagios*/
12        }
13    };

```

Os *traces* adicionados produziram saídas que possibilitam verificar passo-a-passo a execução de cada instrução e a mudanças de estados da CPU. As saída produzidas pela execução da instrução **VMXON** pode ser vista no Algoritmo 6.3. Este *trace* mostra que a instrução **VMXON** executou da seguinte maneira: primeiramente foi feito um teste para verificar se as instruções de virtualização não estão habilitadas; na linha 5 do *trace*, pode-se verificar que durante a execução da instrução **VMXON** o **VMCS** ativo foi inicializado; e por fim, na linha 6, as instruções de virtualização são habilitadas e a instrução **VMXON** cumpre sua função.

Algoritmo 6.3: Saída do *trace* acrescentado à descrição de comportamento da instrução **VMXON**.

```

1 ArchC: _____Starting Simulation _____
2 _____ PC=0 _____ 0
3 vmxon
4 RS_VMX_OP=OFF
5 RS_VMCS_ATIVO=0
6 RS_VMX_OP=ON
7 _____ PC=0x4 _____ 1

```

7 CONCLUSÃO

Esta monografia apresenta a descrição de um modelo de simulador com estruturas e instruções de suporte à virtualização inspirados na tecnologia Intel VT-x sobre um processador MIPS R3000. O intuito é fornecer uma ferramenta que facilite o entendimento e estudo da tecnologia de suporte de hardware à virtualização, mais especificamente da Intel VT-x.

Foi justificada a necessidade de estudo do suporte de hardware à virtualização, pois outros estudos mostram que sistemas virtualizados com técnicas de software atingem maior desempenho quando comparados a sistemas virtualizados com técnicas de hardware.

Também foram apresentadas as principais características da tecnologia Intel VT-x, para suporte de hardware à virtualização em arquiteturas IA-32. Foi apresentado o MIPS-vt, que é o modelo MIPS R3000 estendido com instruções e estruturas inspiradas na tecnologia Intel VT-x para oferecer suporte de hardware à virtualização. Por fim, foi descrito um teste para verificar o correto funcionamento do modelo MIPS-vt e os resultados obtidos.

Conclui-se que é possível mapear tecnologias de suporte de hardware a virtualização, como Intel VT-x, para modelos mais simples de arquiteturas bem conhecidas como o MIPS R3000 e, com isso, gerar simuladores. Isso possibilita um melhor entendimento da virtualização de hardware.

Com os testes executados, além de verificar o correto funcionamento do modelo, foi possível averiguar que o número de instruções de virtualização executadas no MIPS-vt é reduzido em comparação com instruções lógicas e aritméticas. Além disso, através do acréscimo de *traces* foi possível acompanhar passo-a-passo a execução das instruções de virtualização modeladas no MIPS-vt.

Como trabalhos futuros pretende-se comparar a virtualização feita com o MIPS-vt com a virtualização realizadas sem a presença de suporte de hardware. Também pretende-se construir um sistema com múltiplos núcleos de MIPS-vt para avaliar o seu comportamento e possivelmente sugerir instruções ou estruturas específicas para aumentar o desempenho da virtualização de hardware em ambientes com muitos núcleos.

REFERÊNCIAS

- [ADAMS,AGESEN,2006] ADAMS, K.; AGESEN, O. **A Comparison of Software and Hardware Techniques for x86 Virtualization**, Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, California, USA, 2006.
- [AMD,2005] Advanced Micro Devices (AMD) **AMD64 Virtualization Codenamed "Pacifica" Technology - Secure Virtual Machine Architecture Reference Manual**, 2005.
- [ARAÚJO et al,2005] ARAÚJO, C.; GOMES, M.; BARROS, E.; RIGO, S.; AZEVEDO, R.; ARAÚJO, G. **Platform Designer: An Approach for Modeling Multiprocessor Platforms based on SystemC**, Journal of Design Automation for Embedded Systems, Springer, pp.253-283, Vol. 10, Number 4, December 2005.
- [ARCHC,2008] Página do ArchC, Disponível em Disponível em: <<http://www.archc.org/>>. Acesso em: junho de 2008.
- [ARCHC TEAM,2004] The ArchC Team **The ArchC Architecture Description Language Reference Manual**, Computar System Laboratory (LSC) - Institute of Computing, University of Campinas, Disponível em: <<http://www.archc.org/>>, 2004.
- [ASHENDEN,1990] Ashenden, J. P. **The VHDL Cookbook - First Edition**, Dept. Computer Science University of Adelaide South Australia, 1990.
- [BASHFORD et al,1994] BASHFORD, S; LEUPERS, R. et al **The MIMOLA Language, version 4.1, Reference Manual**, Department of Computer Science, University of Dortmund, 1994.
- [CASAZZA et al,2006] Casazza, J. P., Greenfield, M., Shi, K. **Redefining Server Performance Characterization for Virtualization Benchmarking**, Intel Technology Journal, 2006.
- [DONG et al,2006] DONG, Y.; LI, S. et al **Extending Xen with Intel Virtualization Technology**, Intel Technology Journal, v. 10, 2006.
- [FAUTH et al,1995] FAUTH, A.; VAN PRAET, J.; FREERICH, M. **Describing instruction set processors using nML**, Proceedings of the 1995 European conference on Design and Test, p.503, 1995.

- [WOLFFE et al,2002] WOLFFE, G. S. et al **Teaching Computer Organization/Architecture With Limited Resources Using Simulators**, SIGCSE, Kentucky, USA, 2002.
- [HENNESSY et al,1998] HENNESSY, J. L. AND PATTERSON, D. A. **Computer Organization & Design: The Hardware/Software Interface**, Morgan Kaufmann, 1998.
- [INTEL CORP VT-i,2005] INTEL CORPORATION **Intel Virtualization Technology Specification for the Intel Itanium Architecture (VT-i)**, 2005.
- [INTEL CORP VT-x,2005] INTEL CORPORATION **Intel Virtualization Technology Specification for the IA-32 Intel Architecture**, 2005.
- [MIDAS WEB PAGE,2008] Página do MIDAS, Disponível em <<http://cis.stanford.edu/icl/wooley-grp/midas.html>>. Acesso em junho de 2008.
- [MISHRA,DUTT,2005] MISHRA, P.; NIKIL, D. D. **Functional Verification of Programmable Embedded Architectures - Chapter 2, Architecture Specification**, Springer, US, 2005.
- [POPEK,GOLDBERG,1974] POPEK, G. J.; GOLDBERG, R. P. **Formal Requirements for virtualizable third generation architectures**, *Commun, ACM*, vol 17, p. 412-421, 1974.
- [RIGO et al,SBAC-2004] RIGO, S.; ARAÚJO, G.; BARTHOLOMEU, M.; AZEVEDO, R. **ArchC: a SystemC-based Architecture Description Language**, SBAC-PAD, 2004.
- [RIGO et al,2005] RIGO, S.; AZEVEDO, R.; ARAÚJO, G.; CENTODUCATTE, P. **Construção de Modelos de Processadores Usando uma Linguagem de Descrição de Arquiteturas**, VI Workshop em Sistemas Computacionais de Alto Desempenho, 2005.
- [ROSENBLUM,GARFINKEL,2005] ROSENBLUM, M.; GARFINKEL, T. **Virtual Machine Monitors: Current Technology and Future Trends**, *IEEE Computer*, vol. 38, 2005.
- [ROSE,2004] ROSE, R. **Survey of System Virtualization Techniques**, CiteSeer.IST, 2004, Disponível em <<http://citeseer.ist.psu.edu/720518.html>>.
- [SIMPLESCALAR,2008] Página do Simplescalar, Disponível em: <<http://www.simplescalar.com/>>. Acesso em junho de 2008.
- [SYSTEMC,2008] Página do SystemC, Disponível em:<<http://www.systemc.org>>. Acesso em: junho de 2008.
- [SYSTEMC TEAM,2002] SYSTEMC TEAM **Functional Specification for SystemC 2.0**, Disponível em:<<http://www.systemc.org>>. Acesso em: junho de 2008.
- [UHLIG et al,2005] UHLIG, R.; NEIGER, G. et al **Intel Virtualization Technology**, *Computer Journal*, v. 38, p. 48-56, 2005.

[XEN SOURCE,2007] XEN SOURCE A Performance Com-
parison of Commercial Hypervisors, Disponível em
<[http://www.xensource.com/files/hypervisor_performance_comparison_1_0_5_with_esx-
data.pdf](http://www.xensource.com/files/hypervisor_performance_comparison_1_0_5_with_esx-data.pdf)>, acesso em junho de 2008, 2007.

ANEXO DOCUMENTAÇÕES

Neste anexo encontram-se primeiramente relacionadas as publicações feitas, submetidas e aceitas durante o desenvolvimento desta monografia. Posteriormente é indicada a página web onde foram publicados diversos documentos sobre o trabalho realizado durante o desenvolvimento desta monografia.

Artigos

- FERREIRA, M. K.; FREITAS, H. C.; NAVAUX, P. O. A. **From Intel VT-x to MIPS: An ArchC-based Model to Understanding the Hardware Virtualization Support**, WCAE-ISCA, p. 9-15, Beijing, China, 2008.
- FERREIRA, M. K.; FREITAS, H. C.; NAVAUX, P. O. A. **Modificações no MIPS Inspiradas na Intel VTx para Suporte à Virtualização Utilizando ArchC**, Escola Regional de Alto Desempenho, p. 245-248, Santa Cruz - RS, Brasil, 2008.
- FERREIRA, M. K.; FREITAS, H. C.; NAVAUX, P. O. A. **Estudo das Técnicas de Suporte à Virtualização para Projeto de Instruções no Contexto MultiCore**, Workshop de Processamento Paralelo e Distribuído, p. 79-80, Porto Alegre - RS, Brasil, 2007.

Página Web

- Página do Projeto VISA: <http://www.codeplex.com/visa>