Uma plataforma para o ensino de organização de computadores e linguagem de montagem

Gabriel Krisman Bertazi, Rafael Auler, Edson Borin
Instituto de Computação - Unicamp
Campinas, Brasil
gabriel.bertazi@students.ic.unicamp.br, auler@ic.unicamp.br, edson@ic.unicamp.br

***Resumo*—O ensino de disciplinas introdutórias de arquitetura de computadores é essencial para a formação de engenheiros e cientistas da computação. Neste artigo, nós apresentamos uma infraestrutura baseada em um simulador de plataforma de um processador ARMv7 que é utilizada nas atividades práticas da disciplina introdutória de Organização de computadores e Linguagens de montagem da Universidade Estadual de Campinas.**

***Keywords*—Ensino de arquitetura, Linguagens de montagem, Organização de computadores.**

I. Introdução

A compreensão de características arquiteturais dos computadores modernos, bem como o entendimento da interação entre as camadas da pilha de execução, são essenciais à formação de engenheiros e cientistas da computação que almejem construir programas eficientes. Por outro lado, a complexidade das arquiteturas atuais oferece desafios nem sempre triviais à compreensão de alunos de graduação. Neste contexto, cursos introdutórios que apresentam a arquitetura do conjunto de instruções de um processador juntamente com conceitos básicos de organização de computadores são essenciais à formação dos alunos.

Borin e Auler [1] apresentam uma metodologia e plano didático para um curso introdutório de organização de computadores e linguagens de montagem que utiliza simuladores de arquiteturas reais para a realização de exercícios práticos de programação. A utilização de simuladores é vantajosa no contexto educacional por permitir que os estudantes tenham amplo acesso ao estado interno da máquina, de forma que possam observar a interação dos programas desenvolvidos com o sistema, o que facilita a depuração e o estudo da arquitetura modelada.

Este artigo apresenta uma plataforma para o ensino de organização de computadores e linguagens de montagem composta por: (a) um simulador de sistema constituído por um processador ARMv7 [2] e periféricos e (b) ferramentas auxiliares para geração e execução de programas no simulador. O simulador emula o conjunto de instruções de um processador ARMv7 e alguns dos dispositivos periféricos que compõem um *system-on-a-chip* (SoC) desta arquitetura, permitindo aos alunos o estudo e desenvolvimento de programas tanto em nível de usuário quanto em nível de sistema. As ferramentas auxiliares, que incluem desde utilitários de montagem e geração do código até programas que executam no ambiente simulado interagindo com o código do aluno, permitem que ele obtenha uma visão global da pilha de execução de um computador moderno.

A infraestrutura apresentada vem sendo utilizada de forma bem sucedida, desde 2011, no curso introdutório de Organização Básica de Computadores e Linguagens de Montagem, ministrado no 4º e no 5º semestre dos cursos de Engenharia da Computação, Engenharia de Controle e Automação e Ciência da Computação da Universidade Estadual de Campinas. A ferramenta é utilizada nas atividades práticas da disciplina, visando a fixação dos conceitos estudados, e tem recebido diversas avaliações positivas dos alunos.

O restante deste texto está organizado da seguinte forma: a seção II apresenta o simulador; a seção III discute as interfaces para visualização do estado interno do simulador e depuração; a seção IV discute o uso destas ferramentas no ensino de organização de computadores, através do desenvolvimento de aplicações em espaço de usuário e em espaço de sistema; por fim, a seção V apresenta as conclusões.

II. Simulador de plataforma ARMv7

A plataforma simulada foi inspirada em uma plataforma de desenvolvimento real, chamada *Freescale i.MX53 Quick Start Board* [3]. A Figura 1 apresenta uma visão geral dos componentes da plataforma simulada.

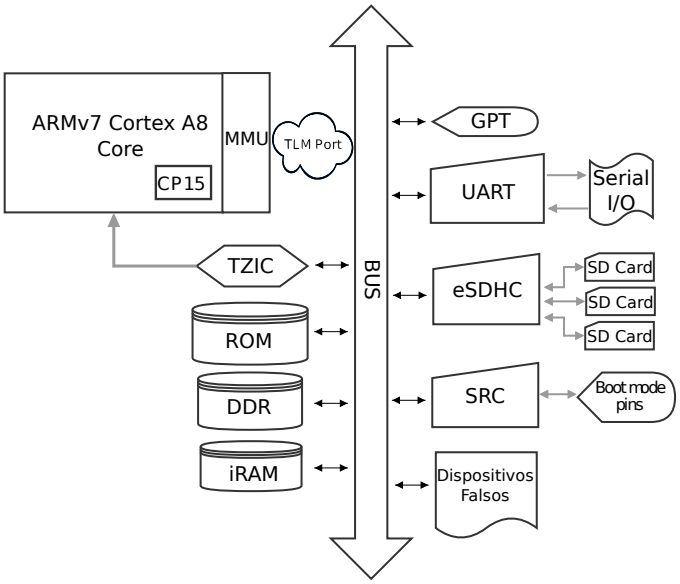

Figura 1. Visão geral dos componentes da plataforma.

Tanto o processador quanto os demais periféricos são conectados a um barramento que gerencia o tráfego de

informações entre os componentes. Além de se conectar ao barramento principal, o dispositivo TZIC, que controla as interrupções, é conectado diretamente ao processador por um barramento exclusivo que permite a geração de interrupções no processador.

Os dispositivos periféricos e o processador foram desenvolvidos de forma a serem fiéis às especificações do *hardware* real. Assim, os programas escritos pelos alunos para o simulador podem ser executados sem alterações na plataforma real.

O componente ARMv7 Cortex-A8, que representa o processador, é o módulo mais complexo do simulador e foi gerado com o auxílio das ferramentas e da linguagem de descrição de arquiteturas ArchC [4]. A partir de uma descrição da arquitetura e do conjunto de instruções em ArchC, a ferramenta `acsim` gera um componente SystemC [5] que simula a unidade central de processamento. Os periféricos do sistema foram implementados manualmente como módulos SystemC e conectados ao processador através de um barramento, também desenvolvido em SystemC.

As principais vantagens de um simulador em relação ao sistema real são:

- O simulador possui uma detalhada saída para depuração que imprime cada aspecto do sistema que o usuário esteja interessado em observar, como transações no barramento, ciclos de relógio e atividade interna dos periféricos e instruções executadas no processador.
- O aluno tem a oportunidade de rodar seu próprio código de sistema ARM que irá configurar e realizar o processo de *boot* da plataforma virtual. Em uma plataforma real, em contraste, este código geralmente é escrito pelo fabricante do SoC em memória somente leitura.
- Graças a um recurso dos simuladores gerados pelo `acsim`, o aluno pode conectar uma instância do depurador GDB diretamente ao simulador e inspecionar, através de uma linha de comando interativa do GDB, cada instrução do sistema, passo a passo, e não só de um processo. Por exemplo, se ocorrer uma interrupção, o aluno terá a oportunidade de acompanhar como um sistema operacional irá lidar com ela, instrução por instrução.

Portanto, a plataforma virtual, ou simulador de sistema, é uma ferramenta que viabiliza o estudo e a proposta de exercícios abrangentes e esclarecedores sobre o funcionamento de um computador real.

A seguir, apresentamos uma explicação individual a respeito de alguns componentes do modelo que podem ser explorados no contexto do curso de organização de computadores.

A. UART

O primeiro periférico disponível na plataforma virtual é a UART[1]. Como todos os outros módulos do sistema, ela realiza uma simulação fiel ao comportamento de uma UART em *hardware* real. O objetivo deste componente é ensinar ao aluno como configurar e usar o dispositivo para a comunicação de entrada no sistema. Ao escrever código em linguagem de montagem para um pequeno *driver* de UART, esperamos que o aluno obtenha consciência de como um sistema computacional pode interagir com o usuário no nível mais elementar possível: lendo e escrevendo em endereços associados a portas de leitura, escrita e controle de dispositivos periféricos.

A entrada e saída da UART é simulada com a leitura da entrada padrão e a escrita na saída padrão do processo correspondente ao simulador, permitindo a inspeção do conteúdo escrito na UART bem como a entrada de dados.

B. GPT

O *General Purpose Timer*, ou temporizador de propósito geral, é um componente que pode ser programado para gerar uma interrupção no processador assim que um determinado número de ciclos (programado pelo usuário) transcorrer desde o início da contagem. Este componente é ideal para que o aluno se acostume com interrupções e absorva a ideia de que o seu programa, em um sistema real, dificilmente irá executar uma sequência de instruções sem ser interrompido e ter seu código intercalado com outros programas. Nos exercícios com o GPT, o aluno também aprende a importância do código do sistema ser transparente ao código de usuário (que é interrompido), sob pena de causar um comportamento difícil de prever.

C. TZIC

O *TrustZone Interrupt Controller* é uma réplica do controlador de interrupções encontrado em SoCs com processadores ARM. Este dispositivo consiste em um pequeno *hardware* capaz de organizar prioridade entre diversos periféricos que interrompem o processador. Ao lidar com o TZIC, o aluno tem uma noção precisa de como o sistema coordena seu funcionamento e do atendimento de diversas interrupções ao mesmo tempo.

D. eSDHC e cartões SD

O eSDHC, *enhanced SD Host Controller*, é um dispositivo que permite fazer leitura e escrita em uma unidade de armazenamento persistente que simula um cartão SD [6]. O eSDHC e o modelo do cartão SD são utilizados para carregar um pequeno sistema operacional e o código do aluno para dentro da memória RAM do simulador, durante a inicialização do modelo.

Este esquema de inicialização permite que o aluno entenda que seu programa não é o único em execução no computador, bem como explica como o seu código foi carregado e começou a executar.

As seções a seguir detalham os mecanismos de depuração implementados, o funcionamento da pilha de execução no nosso simulador e como os programas escritos pelos alunos fazem parte desta pilha.

III. Visão interna do estado da simulação

Uma grande vantagem da utilização de simuladores no ensino de disciplinas de arquitetura de computadores está relacionada à facilidade que estes modelos oferecem para que

[1]Do inglês, *Universal Asynchronous Receiver/Transmitter*.

o usuário possa inspecionar o estado interno da máquina simulada. Diferentemente do dispositivo real, que pode depender de mecanismos de *hardware* adicionais para a depuração, limitando o uso por vários alunos ao mesmo tempo, o simulador oferece diversos mecanismos para observar o estado interno de cada módulo.

A. *Registro de execução (*Log*)*

O simulador oferece um conjunto de opções de linha de comando acessíveis através da opção `-D`, ou `--debug`, para ativar o registro de execução (*Log*) de cada módulo simulado. Todos os módulos exibidos na Figura 1 têm rotinas de depuração que são ativadas com este parâmetro.

Cada módulo emite um conjunto específico de informações no seu *log* de execução, de forma que o usuário possa compreender o estado do dispositivo naquele momento da simulação. Por exemplo, se utilizarmos a opção `--debug=core`, o processador simulado imprimirá, a cada instrução executada, o endereço do *Program Counter*, o nome e os operandos da instrução e os efeitos da instrução na máquina, como mudanças nos registradores e escritas na memória. Por outro lado, se combinarmos o *log* do núcleo com o do barramento, fazendo `--debug=core,bus`, além das informações do processador, serão impressos também o endereço e dados de cada leitura e escrita executadas no barramento.

Um caso especial de informação de depuração é a opção `--debug=flow`, que indenta o *log* e imprime valores de entrada e saída das sub-rotinas do código executado. Desta forma, o estudante pode verificar facilmente se a interface entre suas sub-rotinas está implementada corretamente.

A Figura 2 apresenta um exemplo de saída de depuração utilizando os parâmetros `--debug=flow,core`. Pode-se observar que o modelo imprime a execução de cada instrução, bem como identifica a entrada e saída de sub-rotinas, segundo o padrão de chamadas de função da arquitetura [7].

Apesar do mecanismo de *log* ser útil para depuração de programas pequenos e de periféricos, ele não é prático para programas grandes, que possuem muitas instruções. Neste cenário, o uso de ferramentas mais poderosas, como o depurador GDB, são necessárias.

B. *Interface com o depurador GDB*

O simulador oferece uma interface para conexão com o GDB através dos parâmetros `--gdb` e `--gdb-port`.

Estas opções fazem com que o simulador aguarde uma conexão do depurador através da porta especificada, antes de iniciar a simulação. O estudante pode utilizar uma distribuição oficial do GDB compilada com a opção `--target=arm-linux-gnueabi`, e conectá-la ao modelo através do comando `target remote`.

Com a conexão estabelecida, o aluno tem controle total sobre a simulação, podendo inserir *breakpoints* para parar a execução em pontos arbitrários, inspecionar a memória e registradores dinamicamente e executar seu código passo a passo, assim como faria no seu ambiente local.

```
--------------- PC=0x0 ------------ 0
Instruction: B
Calculated branch destination: 0x20
--------------- PC=0x20 ----------- 1
Instruction: LDR
--------------- PC=0x24 ----------- 2
Instruction: BL
Calculated branch destination: 0x30
    <Entered Function => 0x30
     Return Address   => 0x28 >
    ------------ PC=0x30 ----------- 3
    Instruction: STM
    ------------ PC=0x34 ----------- 5
    Instruction: ADD
    op1(R1) : 0x3   Op2(R2) : 0x2
    R0 <= R1 + R2 = 0x5
    ----------- PC=0x38 ------------ 6
    Instruction: LDM
    <Returning to address 0x28>
--------------- PC=0x28 ------------ 7
Instruction: B
Calculated branch destination: 0x28
```

Figura 2. Trecho da saída de depuração do modelo utilizando os parâmetros `--debug=flow,core`.

IV. Compreensão da pilha de execução e desenvolvimento de códigos em espaço de sistema

O código do aluno não executa sozinho na plataforma simulada. Assim como seria no *hardware* real, um sistema operacional é responsável por oferecer uma interface de chamadas de sistema e abstrair os dispositivos de *hardware* para o programa em nível de usuário, desenvolvido pelo aluno. Assim, o aluno pode utilizar artifícios como escrita na saída padrão através da chamada de sistema *write*, sem se preocupar num primeiro momento em como, de fato, é feita a escrita nos registradores da UART.

O nosso sistema operacional implementa algumas das chamadas de sistema do padrão POSIX. Assim, o mesmo código que executa no simulador pode ser executado sem alterações em um dispositivo com processador ARM real, operando sobre um sistema operacional GNU/Linux, por exemplo.

Também é importante que o estudante consiga observar que o seu programa não surge de maneira automática, pronto para execução, na memória do simulador. De fato, nós buscamos reproduzir o fluxo de carregamento conforme ocorre em um *hardware* real, ilustrado na Figura 3.

O processo de carregamento começa por um código de inicialização, alocado em um módulo de memória ROM no início do mapa de memória. Este código carrega o sistema operacional de um dispositivo de armazenamento persistente que, por sua vez, carrega o código de usuário e inicia sua execução dentro do modo protegido[2] da CPU. O sistema operacional também é responsável por executar as chamadas de sistema, acessadas através de instruções dedicadas pelo código executando em espaço de usuário.

[2]O processador ARM conta com diferentes modos de execução para, entre outras funcionalidades, isolar e limitar programas executando em espaço de usuário.

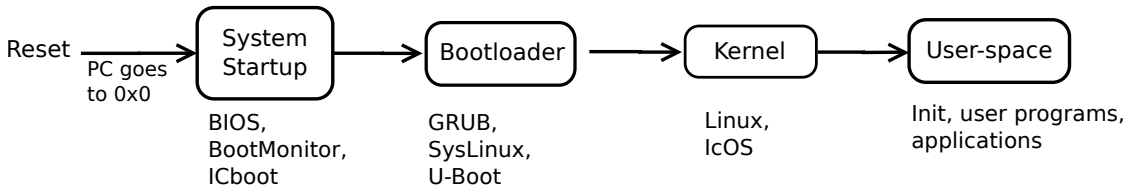

Figura 3. Visão geral do processo de *boot* no modelo.

No simulador, foram desenvolvidos programas que reproduzem as etapas deste processo. O IcBoot é um programa escrito em linguagem de montagem do ARM que é responsável pela etapa inicial do processo de carregamento do simulador. Ele pode ser passado através do parâmetro `--rom=`, sendo carregado na memória ROM simulada, de forma que é o primeiro programa a executar no simulador. Sua tarefa é carregar o sistema operacional de dentro da imagem de cartão SD (fornecida pelo parâmetro `--sd=`), e entregar para ele o controle do fluxo de execução.

O IcOS é o sistema operacional desenvolvido para o simulador. Seu trabalho é preparar o ambiente para a execução do código de usuário, que também é carregado do cartão SD e servir a sua execução, através da interface de chamadas de sistema. A vantagem de usar o IcOS ao invés de um sistema operacional real está na simplicidade e flexibilidade que ele oferece. Ainda, por ser escrito diretamente em linguagem de montagem, os alunos podem se basear diretamente em partes dele para escrever as suas versões de códigos de sistema, o que seria muito mais difícil caso usássemos um núcleo Linux completo.

Todo este processo de inicialização, bem como as interfaces bem definidas entre código de usuário e código em espaço de sistema, oferecem aos alunos uma visão completa da pilha de execução e da necessidade da especificação e implementação correta das interfaces entre estes códigos.

Uma vantagem do simulador frente ao *hardware* real é permitir facilmente que o aluno substitua qualquer etapa do processo de carregamento pelo seu próprio programa, capaz de desempenhar este papel. Assim, o aluno tem a oportunidade de desenvolver seu próprio sistema operacional ou código de inicialização executando em espaço de sistema, algo que seria muito mais complexo de se fazer no dispositivo real.

Assim, o aluno pratica o desenvolvimento de aplicações em espaço de sistema, exercitando muitos conceitos importantes, como a comunicação com periféricos do sistema, gerenciamento e tratamento de interrupções e modos do processador, além de compreender a importância de ter interfaces bem definidas, principalmente ao ter que implementar a interface de chamadas de sistema para o programas executando nos níveis acima da pilha de execução.

V. Conclusão

Este artigo apresentou uma infraestrutura para o ensino de linguagens de montagem e organização básica de computadores utilizando um simulador de plataforma ARMv7. A infraestrutura permite que o estudante pratique o desenvolvimento tanto de programas executando em espaço de usuário quanto em espaço de sistema.

Para o modo de usuário, nós dispomos de um sistema operacional minimalista que serve diversas chamadas de sistema POSIX para o código do aluno, de forma que ele não tenha que se preocupar com detalhes arquiteturais num primeiro momento. O simulador também permite que o aluno implemente seu próprio código de sistema, modo em que o código do aluno será responsável por gerenciar interrupções, acessar periféricos pelo barramento e responder às requisições de programas em espaço de usuário.

O simulador também provê uma ampla estrutura para depuração, permitindo que o estudante tenha uma visão global do sistema, com acesso ao registro de execução de instruções e seus efeitos e ao registro de modificações nos diversos dispositivos periféricos modelados. Por fim, uma interface com o depurador GDB permite que o aluno execute seu programa passo a passo, acompanhando o estado da máquina simulada após cada instrução executada.

O uso da plataforma virtual apresentada neste trabalho se mostrou benéfica ao ensino de linguagens de montagem e organização de computadores no curso de graduação da Unicamp em que foi aplicada, ao permitir a prática dos conceitos estudados em aulas teóricas. A infraestrutura apresentada neste trabalho, bem como materiais adicionais e apostilas didáticas estão disponíveis em http://www.ic.unicamp.br/~edson/disciplinas/mc404/material/.

VI. Agradecimentos

Os autores agradecem ao CNPq e à FAPESP pelo apoio e suporte oferecidos para a realização deste trabalho.

Referências

[1] E. Borin e R. Auler, "Uma abordagem para o ensino de linguagem de montagem, arquitetura e organização de computadores," Em *Workshop sobre Educação em Arquitetura de Computadores (WEAC)*, Ipojuca, Brasil, 2013.

[2] ARM Limited, "Armv7-A Architecture Reference Manual," http://infocenter.arm.com/help/topic/com.arm.doc.ddi0406c, acessado em Agosto de 2014.

[3] Freescale Semiconductors, Inc., *i.MX53 Multimedia Applications Processor Reference Manual*, imx53rm rev 2.1 ed., Arizona - USA, 2012.

[4] S. Rigo, G. Araujo, M. Bartholomeu, e R. Azevedo, "ArchC: a SystemC-based architecture description language," Em *16th Symposium on Computer Architecture and High Performance Computing - SBAC-PAD*, 2004.

[5] IEEE, *1666-2011 INT/2005 Edition, IEEE Standard Interpretations of IEEE Standard Open SystemC Language Reference Manual(IEEE Std 1666-2011)*, 2011.

[6] *SD Specifications part 1: Physical Layer. Simplified Specification*, Version 2.00 ed., SD Group: Matsushita Electric Industrial Co. Ltd. (Panasonic), SanDisk Corporation, Toshiba Corporation and SD Card Association Technical Committee, San Ramon, CA, USA, 2006.

[7] ARM Limited, "Procedure Call Standard for the ARM Architecture," http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf, Acessado em Agosto de 2014.