Uma abordagem para o ensino de linguagem de montagem, arquitetura e organização de computadores

Edson Borin
Instituto de Computação - Unicamp
Campinas, Brasil
edson@ic.unicamp.br

Rafael Auler
Instituto de Computação - Unicamp
Campinas, Brasil
auler@ic.unicamp.br

***Resumo*—Neste artigo nós apresentamos uma abordagem para o ensino de linguagem de montagem, arquitetura e organização de computadores e discutimos a utilização desta abordagem nos cursos de ciência e engenharia da computação da Universidade Estadual de Campinas.**

I. Introdução

A arquitetura do computador é uma das interfaces da pilha de execução de sistemas de computação. Esta interface define um conjunto de estados e operações que podem ser executados pelo *hardware* em questão. Entender como instruir o computador para executar estas operações e como combinar estas operações para realizar o trabalho descrito em programas de alto nível é fundamental para a compreensão da pilha de execução. Assim sendo, a disciplina de arquitetura de computadores está intimamente ligada à programação em linguagem de montagem e à organização de computadores.

Nos cursos de Ciência e Engenharia da Computação da Unicamp, a organização básica de computadores é introduzida juntamente com a linguagem de montagem no segundo ano do curso. Este curso provê uma visão geral sobre a organização de computadores, ensina a programação de computadores em linguagem de montagem e introduz os conceitos de arquitetura de computadores. Este texto apresenta a abordagem desenvolvida pelos autores para o ensino deste curso.

A abordagem de ensino desenvolvida é dividida em três partes. Na primeira parte, o computador do IAS [1], um dos primeiros computadores eletrônicos de propósito geral, é apresentado. Este computador, desenvolvido pelo matemático John von Neumann no Instituto de Estudos Avançados de Princeton (IAS) no início da década de 50, possuía modelos de programação e execução que ainda servem de base para o projeto de computadores modernos [2]. Além disso, a simplicidade deste computador nos permite focar em conceitos básicos, como a organização dos módulos do computador, o modelo de execução, a programação em baixo nível e o procedimento de montagem. Uma vez dominados estes conceitos, compreender a linguagem de montagem e o funcionamento dos computadores modernos se tornam tarefas mais simples.

Na segunda parte, a arquitetura dos processadores ARM [3], presente em bilhões de dispositivos eletrônicos em todo o mundo, é utilizada para o ensino de programação em linguagem de montagem. Nesta etapa, os alunos aprendem a programar em linguagem de montagem, a converter programas em linguagem de alto nível para linguagem de montagem, a ligar o código de programas em linguagem de alto nível com linguagem de montagem e a importância da ABI[1].

A terceira parte apresenta uma visão geral da organização de computadores modernos, incluindo os barramentos do sistema e os dispositivos de entrada e saída, e ensina a programação destes dispositivos em linguagem de montagem. Nesta etapa, também são introduzidos os conceitos de exceções, interrupções e interrupções por *software* e os mesmos são exercitados através de atividades práticas em laboratório.

Este texto está organizado da seguinte forma: As seções II, III e IV apresentam em detalhes cada uma das partes do curso. Por fim, a Seção V apresenta as conclusões.

II. Parte I: O Computador IAS

Para a primeira parte, nós usamos o computador do IAS, chamado aqui apenas de "IAS", para prover uma visão geral sobre a organização e implementação de computadores. A organização deste computador é bem simples, o que facilita a introdução de diversos conceitos básicos. Este computador possui quatro módulos principais: A memória principal, a unidade de controle, a unidade lógica aritmética e o equipamento de entrada e saída. A Figura 1 apresenta a organização detalhada do IAS.

Da mesma forma que nos computadores modernos, a memória principal é utilizada para armazenar tanto as instruções como os dados do programa. De fato, esta foi uma das primeiras arquiteturas a implementar o conceito do programa armazenado, inventado pela equipe que desenvolveu o computador ENIAC [4]. Um detalhe interessante é que esta memória possuía 1024 palavras de 40 *bits* e cada palavra podia armazenar um número de 40 *bits* na representação complemento de 2 ou duas instruções de 20 *bits*.

A unidade lógica e aritmética continha dois registradores arquiteturais (`AC` e `MQ`), visíveis para o programador, e um registrador interno (`MBR`). A unidade de controle do IAS possuía quatro registradores internos (`PC`, `IR`, `IBR` e `MAR`) que não eram visíveis para o programador e um

[1]do inglês: *Application Binary Interface*

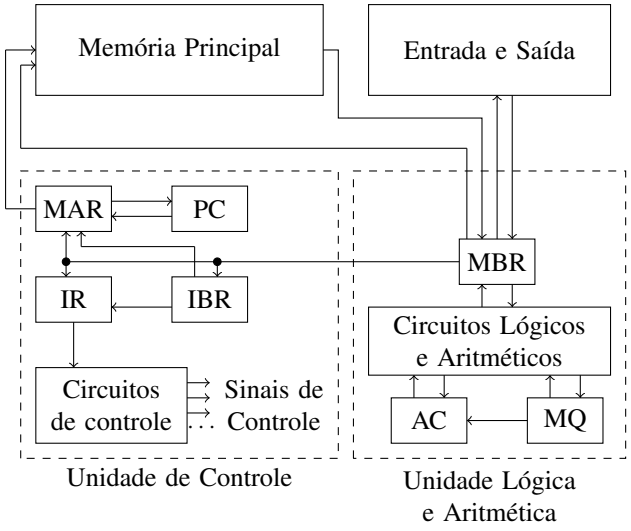

Figura 1. Organização detalhada do computador IAS

pequeno circuito de controle que regia o funcionamento do computador.

A interconexão dos módulos é apresentada como sendo uma forma primitiva de barramento. Nesta etapa, já podemos discutir conceitos básicos de barramentos e subdividí-los em barramentos de dados, endereço e controle.

A estrutura interna e o funcionamento da memória, da unidade de controle e da unidade lógica e aritmética são simples e permitem que cada um dos passos necessários para a execução de instruções seja explicado em detalhes. Este processo permite aos alunos entenderem o papel da memória e dos registradores bem como o funcionamento básico de um computador de propósito geral. Além disso, o discernimento entre os registradores visíveis para o programador e os não visíveis facilita a distinção entre a arquitetura e a implementação de um computador.

Todas as instruções do IAS são organizadas em um único formato, com dois campos. O primeiro campo, de 8 *bits*, armazena o código da operação enquanto o segundo campo, de 12 *bits*, armazena um endereço de memória. Como ambos os campos possuem tamanho múltiplo de 4, podemos identificar facilmente os campos das instruções quando representamos o conteúdo da memória em hexadecimal. Note que o código da operação é representado exatamente com dois dígitos e o campo endereço com três dígitos hexadecimais. Além de facilitar o entendimento do ciclo de execução, a simplicidade do formato permite aos alunos desenvolver pequenos programas diretamente em linguagem de máquina. Nossa experiência indica que introduzir a linguagem de máquina antes da linguagem de montagem ajuda os alunos a distinguirem instruções, dados e diretivas de montagem durante o aprendizado da linguagem de montagem e também ajuda a entender a razão das diretivas.

Uma outra peculiaridade interessante do IAS é que o mesmo não possui um modo de endereçamento de memória indireto. Dessa forma, para acessarmos posições distintas de memória, quando percorremos os elementos de um vetor em um laço, por exemplo, é necessário modificarmos a instrução que acessa a memória atualizando o seu campo endereço. O IAS possui instruções específicas para este propósito e os alunos não apresentaram dificuldade para absorver este conceito. Além disso, quando o modo de endereçamento indireto é apresentado na segunda parte do curso, a sua utilidade já está bem clara para os alunos.

Até este ponto do curso, os alunos já têm uma visão geral da organização básica de um computador, já entendem como e onde são armazenados os dados e as instruções durante a execução do programa, o processo de execução de instruções e a linguagem de máquina. O conteúdo coberto nesta parte é exercitado em atividades de laboratório que incluem a programação, em linguagem de máquina, de um simulador do computador IAS.

Ainda na primeira parte do curso, nós discutimos a dificuldade de se codificar programas em linguagem de máquina e introduzimos uma linguagem de montagem para o computador IAS. Um detalhe interessante deste ponto é que, por causa da experiência com a programação em linguagem de máquina, os alunos rapidamente identificam a necessidade de se associar endereços a instruções e dados na memória e a introdução e o entendimento do conceito de rótulos se tornam simples.

Juntamente com a linguagem de montagem para o IAS, é apresentado o processo de montagem em dois passos, a representação de números na memória e conversão de bases numéricas. Após a introdução destes conceitos, os alunos implementam, como trabalho prático, um montador para a linguagem de montagem apresentada. Este trabalho ainda é implementado em uma linguagem de alto nível, como C.

Ao término da primeira parte do curso os alunos já: a) têm uma visão geral da organização básica de um computador, b) entendem como as instruções e os dados são representados e armazenados durante a execução de programas, c) entendem o processo de execução de instruções e como o fluxo de execução pode ser controlado por *software*, d) dominam a linguagem de máquina e a linguagem de montagem do computador IAS, e e) entendem como o processo de montagem em dois passos funciona. Nas últimas 4 edições deste curso, foram utilizadas aproximadamente 4 aulas teóricas e 5 aulas práticas para esta parte do curso. Além do montador, entregue como trabalho prático, o aprendizado desta etapa do curso é realizado através de uma prova teórica.

III. Parte II: Linguagem de montagem do ARM

Uma vez que os conceitos básicos foram assimilados, os alunos estão preparados para entender o conjunto de instruções de uma arquitetura mais sofisticada bem como as convenções e mecanismos utilizados para transformar programas em linguagem de alto nível em linguagem de montagem.

Esta parte do curso se inicia com uma introdução a memórias modernas, endereçadas a *byte*, à representação de cadeias de caracteres e de números na memória. É discutido como números com mais de 8 *bits* podem ser

armazenados neste tipo de memória e são apresentados os formatos *little-endian* e *big-endian*. Assim que a memória e a representação dos dados na memória são apresentadas, a arquitetura do conjunto de instruções do processador pode ser introduzida. A arquitetura utilizada na segunda parte do curso é a arquitetura ARMv7 [5], uma variação recente da arquitetura que conquistou o segmento de dispositivos móveis.

A primeira parte da arquitetura a ser apresentada é o conjunto de registradores. Nesta parte do curso, apenas os registradores visíveis no modo usuário são apresentados. Graças à introdução com o IAS, os alunos assimilam mais facilmente o propósito dos registradores e o conceito de arquiteturas *load*/*store*. Uma vez que estes conceitos estão claros, as instruções lógicas e aritméticas são apresentadas. Os diversos tipos de operandos e modificadores presentes nas instruções lógicas e aritméticas também são apresentados. Nesta parte abre-se espaço para uma discussão sobre a limitação dos operandos imediatos em função do espaço disponível para codificação dos campos na instrução, um tópico importante em arquiteturas de computadores.

A introdução de somas e subtração de números de 32 *bits* fomenta a discussão sobre transbordamento (*overflow*), *carry-in*, *carry-out* e operações de soma e subtração de números maiores do que 32 *bits* com instruções que levam em consideração a *flag* de *carry*. Como a informação de transbordamento e de *carry* são armazenadas como *flags* no registrador `CPSR`, este é um ponto natural para introdução das outras *flags* do processador. Assim que o conceito de transbordamento e as *flags* são introduzidos, podemos apresentar o mecanismo de comparação de números no processador. Começamos com uma visão simplificada do processo, analisando o sinal do resultado da subtração de números, e depois elaboramos a comparação com a presença de transbordamento. A comparação com o tratamento de transbordamento requer que múltiplas *flags* sejam inspecionadas para se comparar dois números. Entretanto, os códigos de condição (p. ex. `LT` e `GE`), introduzidos na sequência, simplificam o processo de comparação e permitem que os alunos abstraiam o mecanismo de comparação das *flags* e concentrem sua atenção na lógica do programa durante a programação em linguagem de montagem.

Após a introdução das instruções `STM` e `LDM` para transferência de dados entre a memória e os registradores, introduzimos os conceitos e mecanismos utilizados na implementação de subrotinas. Primeiramente, apresentamos as instruções de chamada e retorno de funções. Diferente das arquiteturas da família x86 da Intel, a instrução de chamada de rotina da arquitetura ARM não empilha o endereço de retorno automaticamente na pilha do programa. Este endereço é salvo no registrador `R14` e cabe ao programador empilhar o seu valor, caso necessário. Esta distinção permite que os alunos façam rotinas que não chamam outras rotinas, chamadas rotinas "folhas", sem que a pilha seja necessária. A introdução das instruções de chamada e retorno trazem à tona a discussão sobre onde devemos salvar o valor do registrador `R14` quando implementamos rotinas que chamam outras rotinas, ou rotinas recursivas. Neste ponto, introduzimos a pilha do programa, o apontador da pilha e as operações de empilhamento e desempilhamento.

Após apresentarmos os mecanismos de passagem de parâmetros, discutimos a política de uso de registradores na arquitetura. Mais especificamente, discutimos quais registradores devem ser salvos pela rotina chamadora (*caller-save*) e quais devem ser salvos pela rotina que é chamada (*callee-save*). Esta informação induz a discussão sobre quem é responsável pelo quê e permite que o código em linguagem de montagem gerado pelos alunos seja ligado ao código produzido pelo compilador e executado de forma correta. A passagem de parâmetros por valor e por referência e o retorno de valores em chamadas de funções também são apresentados neste ponto.

Ainda nesta parte do curso, apresentamos a alocação, desalocação e acesso às variáveis locais. Estas variáveis, alocadas na pilha do programa, não são associadas a rótulos e o acesso às mesmas é realizado através de deslocamentos para apontadores de pilha. Os exemplos apresentados mostram que a alocação de dados na pilha pode fazer com que o apontador de pilha, ou *stack pointer*, aponte para uma posição de memória longe dos parâmetros da função e torne o acesso a estes dados um problema. Neste momento, surge a motivação para a introdução do apontador de quadro, ou *frame pointer*, e o conceito de quadro da função é apresentado.

Finalmente, é apresentado o mecanismo de chamada de sistemas, que permite invocar o sistema operacional para realização de serviços como entrada e saída de dados.

Ao fim desta parte do curso, os alunos devem ter aprendido a arquitetura do conjunto de instruções dos processadores ARM e a programar estes processadores em linguagem de montagem. Uma das opções utilizadas para exercitar os conceitos teóricos e avaliar o aprendizado é a realização de um trabalho prático. Os trabalhos aplicados nas últimas edições do curso consistem na implementação de um simulador para o computador IAS em linguagem de montagem ARM ou na implementação das rotinas `scanf` e `printf` da biblioteca GNU `libc`.

IV. Parte III: Código de Sistema e Programação de Periféricos

A terceira parte do curso apresenta os conceitos de programação de periféricos, interrupções, interrupções por *software*, exceções, e os modos de execução do processador. Iniciamos com uma visão geral sobre barramentos e discutimos a conexão física entre periféricos, o processador e a memória. Ainda nesta parte, discutimos o conceito de portas e como o *software* acessa os periféricos através de instruções especiais e através de instruções de acesso à memória. No segundo caso, é explicado o mapeamento de endereços em dispositivos físicos e o conceito de entradas e saídas mapeadas em memória.

A comunicação com os dispositivos periféricos nos processadores ARM é feita através de instruções de acesso à memória. Este conceito é apresentado utilizando-se

exemplos simples, como o exemplo do elevador. Neste exemplo, o programador tem à sua disposição um sensor que fornece o andar atual de um elevador, conectado ao sistema como um periférico, e um mostrador digital, que permite ao programador ligar e desligar um conjunto de segmentos luminosos para indicar o andar atual, também conectado ao sistema como um periférico. Neste exemplo, os alunos têm que implementar uma rotina chamada `atualiza_andar` que lê o andar atual do sensor e configura o mostrador digital para mostrar o andar atual. Uma das questões levantadas durante o exercício é a frequência com que esta função deve ser chamada. As vantagens e desvantagens de se invocar a função com uma frequência alta ou baixa são discutidas e rapidamente os alunos convergem para a conclusão de que o ideal seria que toda vez que o sensor detectasse uma mudança de andar, o mesmo avisasse o processador, para que este pudesse chamar a função `atualiza_andar`. Este é o ponto em que apresentamos o conceito de interrupções.

A introdução do conceito de interrupção levanta diversas questões: a) como o processador é avisado? b) o que acontece com o código que estava executando no momento em que o processador foi interrompido, c) como o processador sabe onde encontrar a rotina que tratará a interrupção? Estas perguntas tornam a introdução dos conceitos como Vetor de Interrupções, gravação e recuperação do contexto do programa e os modos de operação do processador um processo natural.

Os conceitos teóricos desta parte do curso são fixados através de um trabalho prático em que os alunos têm que implementar um escalonador de tarefas preemptivo. Para tanto, os mesmos têm que produzir código de sistema que:

- inicia os periféricos do sistema, como o controlador de interrupções, o temporizador de propósito geral, ou GPT [2], e o dispositivo de comunicação serial, ou UART.
- invoca um programa de usuário (fornecido pelo professor) após iniciar o sistema.
- implementa as chamadas de sistemas `fork` e `exit` para permitir a criação de término de novos processos.
- trata as interrupções do GPT, trocando o contexto dos processos ativos.
- implementa a chamada de sistema `write` e implementa o tratamento da mesma através da cópia dos *bytes* do *buffer* (passado como parâmetro) para o dispositivo de comunicação serial, a UART.

As chamadas de sistema citadas são uma versão bastante simplificada daquelas encontradas em um sistema operacional Linux [6]. Ao término da terceira parte do curso os alunos: a) têm uma visão geral e experiência prática com os mecanismos de comunicação e controle de periféricos, b) entendem como os mecanismos de interrupção, exceção e interrupção por *software* funcionam, c) diferem e entendem a utilidade dos múltiplos modos de operação do processador e d) possuem uma visão geral de como um sistema operacional pode utilizar estes recursos para proteger o sistema contra a execução de código incorreto ou malicioso e para abstrair os detalhes dos periféricos dos programas de usuários. Nas últimas 4 edições deste curso, foram utilizadas aproximadamente 4 aulas teóricas e 5 aulas práticas nesta etapa. As aulas práticas são utilizadas para realização de uma atividade introdutória à infraestrutura e para acompanhamento do projeto e implementação do trabalho. Além do escalonador de tarefas preemptivo, entregue como trabalho prático, o aprendizado desta etapa do curso é realizado através de uma prova teórica.

[2]do inglês: *General Purpose Timer*

V. Conclusões

Apresentamos neste texto uma abordagem com aulas teóricas e práticas para o ensino de linguagem de montagem, arquitetura e organização de computadores. Nossa experiência mostra que é possível ministrar o curso em um semestre, com duas aulas por semana, sendo uma teórica e uma prática. Além da abordagem, os autores escreveram uma apostila que apresenta a organização e o funcionamento do computador IAS, suas instruções e uma breve introdução à sua programação. A apostila, o material utilizado e o calendário de atividades realizadas na última edição do curso podem ser acessados em: http://www.ic.unicamp.br/~edson/disciplinas/mc404/2013-1s/e/index.html

Agradecimentos

Alguns do exemplos utilizados no curso foram inspirados em materiais compartilhados por outros professores do Instituto de Computação. Gostaríamos de agradecer em particular aos professores Ricardo Anido, Paulo Centoducatte e Célio Guimarães. Agradecemos também aos alunos que forneceram subsídio para melhorar a abordagem. Parte da abordagem desenvolvida foi moldada em função das observações realizadas por estes alunos.

Referências

[1] Institute for Advanced Study, "Electronic Computer Project," http://www.ias.edu/people/vonneumann/ecp, acessado agosto 2013.

[2] D. A. Godse and G. A. P., *Computer Organization*. Technical Publications, 2010.

[3] S. Furber, *ARM System-on-Chip Architecture*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[4] R. Rojas and U. Hashagen, Eds., *The First Computers: History and Architectures*. MIT Press, 2000.

[5] ARM Limited, "ARMv7-M Architecture Reference Manual," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403c/index.html, acessado agosto 2013.

[6] D. Bovet and M. Cesati, *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.