cMIPS – uma Ferramenta Pedagógica para o Estudo de Arquitetura

Roberto A Hexsel & Renato Carmo
Depto de Informática – Universidade Federal do Paraná
Caixa Postal 19.081, 81531-990, Curitiba, PR, Brasil
{roberto,renato}@inf.ufpr.br

***Resumo*—Apresentamos o *cMIPS*, nossa implementação em VHDL do processador MIPS. Seu uso é proposto como plataforma integradora para o ensino de Arquitetura de Computadores e disciplinas correlatas. Mostramos como a utilização da plataforma completa, que inclui o processador e alguns periféricos simples, pode facilitar a apropriação dos conceitos centrais e dos seus interrelacionamentos. O *cMIPS* é um modelo do *pipeline* de cinco segmentos do conjunto de instruções MIPS32.**

I. Introdução

Se escuto, esqueço; se vejo, entendo; se faço, aprendo.

O ditado acima é atribuído a Confúcio e expressa uma ideia que é familiar a todos os envolvidos com Educação, seja a formal seja a informal.

O sucesso da aplicação da ideia do "fazer para aprender" na Arquitetura de Computadores (AC) e disciplinas correlatas está, em boa medida, condicionado à disponibilidade de ferramentas que permitam ao aluno praticar o que é estudado. Neste sentido é indispensável que a experiência do aluno possa ir além do projeto 'implementado' com lápis e papel, mas sim, chegar ao ponto em que seja possível projetar, implementar e, por sobre este projeto implementado, escrever código e executá-lo; modificar características do projeto e verificar o impacto destas modificações etc.

Apresentamos a implementação de um simulador realista do processador MIPS chamado *cMIPS*, porque está organizado como a implementação clássica do processador segmentado em cinco estágios tal como descrito em [1]. Escrito em VHDL, o cMIPS é uma implementação completa do conjunto de instruções MIPS32r2 (cfe. [2]). Sendo uma implementação completa do conjunto de instruções, o cMIPS foi concebido para ser o ambiente de experimentação (e de referência) para as disciplinas correlatas à arquitetura de computadores, desde as que estudam os níveis mais elementares de projeto de *hardware* até as que tem seu foco na construção de *software* básico e sistemas operacionais.

O texto está organizado da seguinte forma. A Seção II apresenta uma sequência de trabalhos que podem ser desenvolvidos empregando o cMIPS como plataforma e aponta alguns dos possíveis ganhos, sob um viés pedagógico. A Seção III contém uma breve discussão sobre simuladores de processadores que poderiam ser empregados no ensino de AC. A Seção IV descreve o projeto e implementação do cMIPS. Nossas conclusões, e direções para trabalhos futuros estão na Seção V. O código fonte do cMIPS está disponível na Sourceforge, em https://sourceforge.net/projects/cmips.

II. cMIPS como Ferramenta de Aprendizado

No que se segue, consideramos um curso de Ciência da Computação, ou de Engenharia de Computação com um conjunto de disciplinas semestrais organizado da seguinte forma:

(a) em Circuitos Digitais são introduzidos portas lógicas, *flip-flops*, contadores;
(b) em Microprocessadores são estudados componentes complexos como a ULA, bloco de registradores; a programação em *assembly* é apresentada como uma API para o *hardware*, e então o circuito que implementa esta API é construído;
(c) em Arquitetura são estudados segmentação, hierarquia de memória, subsistema de E/S;
(d) em Software Básico são estudadas algumas das interfaces entre o *hardware* e o *software* tais como montadores e ligadores, e *drivers* de dispositivos de E/S.
(e) em Sistemas Operacionais (SO) são estudadas as abstrações que tornam o próprio SO uma API para o *hardware*, tais como escalonamento dos recursos, *drivers*, memória virtual.

Num dos possíveis modos de emprego, mais centrado em *hardware*, uma linguagem de modelagem de *hardware* (HDL) deve ser introduzida em Circuitos e/ou em Microprocessadores –voltaremos a este assunto adiante. Na disciplina de Microprocessadores, o cMIPS pode ser usado como plataforma de testes para dispositivos de E/S simples projetados pelos alunos. Dentre as inúmeras possibilidades estão portas paralelas, contadores e temporizadores, interfaces seriais (RS232, USB). Os alunos projetam o dispositivo, o acoplam ao barramento do cMIPS e executam código C ou *assembly* para excitar e testar os modelos.

Na disciplina de Arquitetura o cMIPS pode ser aplicado de muitas formas, que incluem: (i) acompanhar a execução de uma sequência de instruções no processador segmentado –acompanhar o progresso das instruções pelos estágios é uma experiência algo mais realista do que aquela em papel, especialmente considerando-se intertravamentos e adiantamento de valores; (ii) na depuração de desempenho de programas, quando investiga-se o ganho de desempenho se as bolhas causadas por desvios forem todas escondidas nos *branch delay slots*, por exemplo; (iii) acrescentar instruções de multimídia ou ponto

flutuante ao processador; (iv) modificar a hierarquia de memória do sistema, alterando-se o projeto das caches e TLB; e (v) acrescentar periféricos mais sofisticados ao sistema, e/ou controladores de acesso direto à memória. A implementação, em VHDL, do modelo de um processador completo é uma tarefa por demais complexa para um único semestre, na primeira metade do curso.

Em Sistemas Operacionais o modelo pode ser empregado como plataforma de *hardware* para a implementação do esqueleto de sistemas operacionais, ou como plataforma para sistemas operacionais completos tais como o Xinu [3]. O porte de um SO para um processador obrigaria os alunos a: (i) estudar e programar os registradores de controle do processador; (ii) programar o sistema de interrupções; e (iii) escrever em *assembly* os trechos críticos do SO, tais como a rotina de troca de contexto. Do ponto de vista pedagógico, a enorme complexidade de um sistema "de verdade" como o Linux, pode ser um sério obstáculo a sua aplicação como caso de estudo numa primeira disciplina de SO.

Em Software Básico os alunos teriam a chance de depurar, num processador que põe à mostra todos os seus registradores, programas como montadores e *drivers* para dispositivos. Os modelos dos dispositivos poderiam ser tanto os desenvolvidos pelos próprios alunos em outras disciplinas, como poderiam ser modelos 'importados' de repositórios como OpenCores.

A. Nem tudo são flores

Até aqui, discussão pressupõe um mundo ideal, ao menos o que seria o ideal para os docentes da área de AC. Em nossa opinião, vários fatores podem interferir com a efetivação desta proposta no mundo real.

Da escolha do MIPS: O processador MIPS é o processador descrito em detalhes num dos textos mais populares no ensino de AC [1], e há farta documentação sobre o conjunto de instruções [4], [2], [5], bem como compilador e demais ferramentas, de ótima qualidade, em software livre [6], [7]. O sistema de interrupções do MIPS é simples o bastante a ponto de ser compreensível para alunos de terceiro ou quarto períodos.

A motivação principal para a adoção do MIPS é de ordem pedagógica: pretende-se ensinar a programação em *assembly* para alunos que são iniciantes em programação e é importante que a linguagem seja a mais simples e regular possível. É nossa opinião que empregar o 80x86 como veículo para o ensino de *assembly* é uma péssima escolha porque a linguagem é complexa e altamente idiossincrática. Presta-se um enorme desserviço aos alunos forçando-os a programar em uma linguagem complexa e irregular porque desta experiência restará a ideia de que APIs contorcidas são algo aceitável, pois "foi o professor de *assembly* quem deu o (péssimo e desnecessário) exemplo". Portanto, ao estudar o *assembly* regular do MIPS, o aluno fica submetido a um menor nível de "ruído intelectual" do que se for exposto ao 80x86, cujo *design* é fruto de uma longa sequencia de remendos aplicados sobre um projeto que já nasceu como um Frankenstein. Não é isso que se deseja ensinar em Computação, quando alternativas melhores estão disponíveis.

Um argumento que se poderia apresentar contra a utilização do MIPS refere-se "ao mercado". Nossa função de educadores não é preparar mão de obra para o mercado, mas prover aos alunos o conhecimento fundamental que lhes permita apropriar-se de qualquer tecnologia com a qual venham a se defrontar. Em particular, em número de unidades, processadores de arquitetura 80x86 ocupam um nicho de mercado relativamente pequeno (uns poucos pontos percentuais), enquanto que os processadores ARM são os campeões de vendas por uma larga margem. As arquiteturas de MIPS e ARM tem muito mais similaridades do que o 80x86 com qualquer destas duas.

Da programação em VHDL: No início do curso o aluno deve: (i) se defrontar com a experiência mais ou menos traumática da programação imperativa em C ou Pascal; (ii) aprender a manejar uma interface de caractere (alguma *shell* tal como Bash); (iii) no segundo ou terceiro semestre do curso inicia-se em *assembly*; e (iv) nossa proposta envolve ainda VHDL, que embute um paradigma de programação distinto da programação imperativa. Podemos especular até que ponto os alunos conseguem absorver e apreender tanto conteúdo – nossas observações são insuficientes para oferecermos um veredito. Uma forma de minimizar o custo e o impacto intelectual do aprendizado de VHDL seria eliminar a programação em VHDL das disciplinas do início do curso, e introduzi-lo somente em Arquitetura. Por outro lado, também nos parece viável a utilização do cMIPS como uma plataforma de hardware, sem que seja realmente necessário aos alunos entender o código do modelo VHDL. O mesmo não se pode dizer quanto aos docentes.

III. Trabalhos Relacionados

O conjunto de disciplinas da área de Sistemas Computacionais do Bacharelado em Ciência da Computação da UFPR é apresentado em [8], e algumas das disciplinas mencionadas na Seção II são descritas naquele artigo. Todas as ferramentas mencionadas aqui foram empregadas em disciplinas por, pelo menos, um dos autores.

O modelo *MRStd* [9] contém uma descrição parcial da implementação com "ciclo longo", no qual todas as instruções demoram um ciclo longo para executar. Este modelo já foi empregado com sucesso na disciplina de Microprocessadores –os alunos recebem a implementação parcial do conjunto de instruções e devem completá-la para que o processador execute um programa escrito em *assembly*, como o fatorial recursivo, por exemplo. Como este modelo não é uma representação fiel do *pipeline*, só é possível executar código C se forem introduzidas modificações no código gerado pelo compilador, por causa dos *branch delay slots* e *load delay slots*, que são preenchidos pelo compilador, e não são tratados no modelo.

O modelo *miniMIPS* [10] contém uma implementação incompleta da versão segmentada do processador. Por exemplo, faltam as instruções que escrevem e leem bytes da memória, e a instrução de divisão, o que dificulta

sobremaneira o uso de código compilado que emprega *strings*. O previsor de desvios não opera corretamente e o código contém idiossincrasias linguísticas que dificultam sua leitura. Por exemplo, no previsor de desvios são empregados os sinais adresse e adressee para o que poderia ser chamado de address e target. Estes e outros problemas são relatados em [11]

No repositório OpenCores estão disponíveis vários modelos de processadores, inclusive do MIPS. O modelo que mais se assemelha ao cMIPS é o *MIPS Release 1* [12]; este modelo é escrito em Verilog. *Plasma* [13] é um modelo que implementa o *Release 1* do conjunto de instruções e sua organização não segue aquela descrita em [1].

Os modelos mencionados acima são todos sintetizáveis. Dentre os simuladores de arquitetura conhecidos pelos autores o Simplescalar [14] é demasiadamente complexo para curso introdutório e não explicita a implementação do processador. Os simuladores produzidos com ArchC [15] são simuladores funcionais e não permitem escrever descrições detalhadas da implementação.

IV. Projeto e Implementação do cMIPS

A Figura 1 mostra um diagrama de blocos com o sistema modelado no *testbench*, que contém um computador completo, com o processador, caches de instruções e de dados, memórias ROM e RAM, e quatro 'periféricos'.

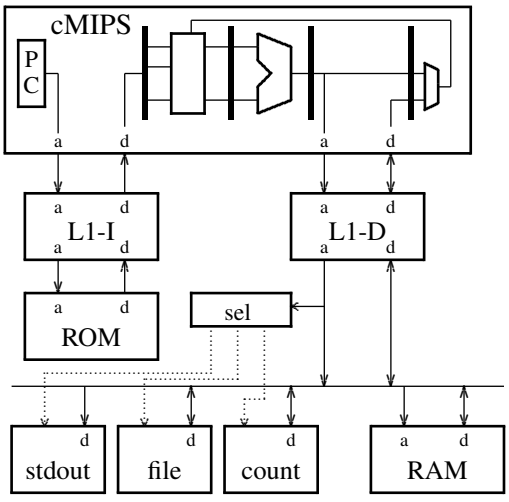

Figura 1. Diagrama de blocos do **testbench**.

O processador contém o circuito de dados de inteiros, e o Coprocessador 0. Além do processador, há modelos para caches de instruções e de dados – ambas são com mapeamento direto, a cache de dados é com escrita forçada (*write-through*) sem alocação de blocos nas faltas por escrita. Acessos a dados podem ocorrer à palavras, meias-palavras, e à bytes. Todos os parâmetros de projeto são derivados da capacidade e do tamanho do bloco. Se não se deseja simular as caches, pode-se instanciar as *fake-caches* que somente repassam dos sinais entre *core* e memórias. As memórias ROM e RAM são inicializadas a partir de arquivos gerados pelos *scripts* de compilação. Estas memórias são carregadas na inicialização do processador, com o sinal *reset* ativo.

Os três periféricos permitem a interação entre o sistema simulado e o mundo externo. O periférico indicado como *stdout* no diagrama permite escrever um inteiro na saída padrão do simulador. O periférico indicado como *file* possibilita a leitura de dados de entrada, e a gravação de resultados, em arquivo. O periférico *count* pode ser programado para gerar uma interrupção após um determinado número de ciclos do relógio.

A. Circuito de Dados

O circuito de dados implementa as instruções de inteiros do conjunto MIPS32r2, com a exceção dos acessos à memória desalinhados porque, aparentemente, estas instruções são patenteadas e não podem ser reproduzidas sem o consentimento do proprietário da patente. O modelo contém todos os circuitos de intertravamento e de adiantamento necessários para a execução correta e eficiente de código compilado.

B. Coprocessor 0

O *processador de controle*, ou Coprocessador 0 (COP0) é implementado parcialmente: as seis interrupções de *hardware* e a interrupção não-mascarável são implementadas no *Interrupt Compatibility Mode*. Os recursos do COP0 são implementados como especificado em [5]. O modelo contém toda a funcionalidade para suportar um sistema operacional *Unix-like*, a menos de um modelo para a TLB, que ainda não está disponível. As instruções de controle do processador, mais as instruções para acessos atômicos, estão implementadas nos registradores de controle descritos resumidamente no que se segue.

O registrador STATUS determina o modo de execução (sistema/usuário), se as interrupções estão habilitadas, e controla outras funcionalidades do processador. O registrador CAUSE indica a causa da excessão ou interrupção. O registrador EPC contém o endereço da instrução que seria executada, não fosse a ocorrência de uma excessão ou interrupção. O registrador CONFIG mantém alguns parâmetros de configuração das caches e do processador. O registrador LLaddr mantém o endereço efetivo do último ll (*load-linked*) executado.

C. Testes e Exemplos

Ao longo do desenvolvimento do cMIPS, um extenso conjunto de programas de teste foi escrito, seja para testar uma ou mais instruções, seja para verificar o modelo através da execução de programas reais. Os programas de teste incluem seis programas de ordenação, cálculo do fatorial, cálculo da Série de Fibonacci, e programas dos CommBench [16] e MiBench [17] (`adpcm`, `crc32`, `drr`, `frag`, `dijkstra` e `stringsearch`) adaptados para execução sem as funções da `libC`. Os programas em *assembly* podem ser usados como exemplos da programação com as instruções menos comuns, ou de uso do COP0.

Os programas de teste em *assembly* foram escritos para testar instruções individuais ou sequências de instruções. São mais de 50 programas de umas poucas instruções, e quando da verificação do modelo, sua execução é seguida num diagrama de tempos gerado com `gtkwawe`.

Os 20 programas em C são programas com tamanho desde umas poucas até 200 linhas de código fonte, que é compilado com otimização `-O2` para os testes.

A cada modificação no modelo todos os programas de teste são executados no processador e a saída é comparada com o que se considera a saída correta. O *script* que executa os testes altera os tempos de acesso de RAM e ROM para verificar se as restrições na temporização dos eventos internos ao processador não foram violadas – são testados 72 combinações de latência de RAM e ROM.

A compilação dos programas de teste é gerenciada por um *script* que compila e liga os programas na ordem correta. Por exemplo, os programas em C devem ser ligados ao programa `start.s`, que inicializa os registradores de controle e a pilha do processador. O mapa de memória é ajustado às configurações de capacidade e de endereços das memórias ROM e RAM. Para executar um programa em C no simulador são necessários somente dois comandos: `compile.sh -ON prog.c`, que compila o programa com nível de otimização `N` e cria os arquivos com código e dados que são lidos pelo simulador, e `run.sh` que compila o modelo VHDL e inicia a simulação. GHDL [18] é empregado para gerar os simuladores a partir do código VHDL.

D. Síntese

Em trabalho preliminar, os resultados da síntese do cMIPS num FPGA Xilinx Spartan-6 (xc6slx16), sem otimização, indicam a utilização de 20% das LUT's daquele dispositivo para processador, memórias e periféricos.

V. Conclusão e Trabalhos Futuros

Este texto apresenta uma proposta de integração das disciplinas da área de Arquitetura de Computadores através de trabalhos práticos que empregam uma mesma ferramenta ao longo de três ou quatro semestres. A ferramenta é um simulador do processador MIPS de 32 bits similar ao descrito no texto de Patterson & Hennessy, codificado em VHDL. A proposta se baseia na ideia de que aprendizado efetivo decorre da prática e a utilização de uma mesma ferramenta numa sequência de duas a quatro disciplinas semestrais ameniza a sua curva de aprendizagem e propicia uma melhor integração dos programas das disciplinas.

Esta proposta pedagógica será posta em prática em nosso Bacharelado nos próximos semestres. Quando coletarmos informações confiáveis sobre os eventuais ganhos para os alunos, estas serão apresentadas neste fórum.

Dentre as tarefas programadas para o futuro próximo está a modelagem da TLB, e a inclusão do suporte completo à memória virtual ao modelo do processador; isso feito, planejamos portar o XINU para o cMIPS.

Referências

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, 4th ed. Morgan Kaufmann, 2009, iSBN 9780123744937.

[2] M. Technologies, "MIPS32 architecture for programmers, volume II: The MIPS32 instruction set," MIPS Technologies, Inc, Rev. 2.50, 2005.

[3] D. E. Comer, *Operating System Design – The XINU Approach*. Prentice-Hall, 1988, vol. 1, iSBN 0136381804.

[4] D. Sweetman, *See MIPS Run – Linux*, 2nd ed. Morgan Kaufmann, 2007, iSBN 0120884216.

[5] M. Technologies, "MIPS32 architecture for programmers, volume III: The MIPS32 privileged resource architecture," MIPS Technologies, Inc, Rev. 2.50, 2005.

[6] R. Stallman *et al.*, "GCC, the GNU Compiler Collection," Free Software Foundation, Páginas html, 1999, disponível em ⟨http://gcc.gnu.org⟩. Acesso em 28/7/2013.

[7] ——, "GNU Binutils," Free Software Foundation, Páginas html, 1998, disponível em ⟨http://ftp.gnu.org/gnu/binutils⟩. Acesso em 28/7/2013.

[8] R. A. Hexsel and R. Carmo, "Ensino de Arquitetura de Computadores com enfoque na interface hardware/software," in *WEAC'06: Workshop sobre Educação em Arquitetura de Computadores*, 2006, pp. 9–16.

[9] N. Calazans and F. Moraes, "Simulação VHDL do processador MRStd," Faculdade de Informática, PUC-RS, Páginas html, 2006, disponível em ⟨http://www.inf.pucrs.br/~calazans/undergrad/orgcomp_EC/mips_mono/mips_v0.vhd⟩. Acesso em 8/5/2012.

[10] S. Hangouët, S. Jan, L.-M. Mouton, and O. Schneider, "miniMIPS," Opencores.org, Páginas html, 2009, disponível em ⟨http://opencores.org/project,minimips⟩. Acesso em 02/08/2013.

[11] J. T. Júnior and R. A. Hexsel, "MPSoC minimalista com caches coerentes implementado num FPGA," in *WSCAD-SSC'09: X Workshop em Sistemas Computacionais de Alto Desempenho*, Oct 2009, pp. 1–8.

[12] G. Ayers, "MIPS32 Release 1," Opencores.org, Páginas html, 2012, disponível em ⟨http://opencores.org/project,mips32r1⟩. Acesso em 02/08/2013.

[13] S. Rhoads, "Plasma - most MIPS I opcodes," Opencores.org, Páginas html, 2001, disponível em ⟨http://opencores.org/project,plasma⟩. Acesso em 21/09/2013.

[14] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[15] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, and E. Barros, "The ArchC architecture description language and tools," *Int Journal of Parallel Programming*, vol. 33, no. 5, pp. 453–484, Oct 2005.

[16] T. Wolf and M. A. Franklin, "CommBench – a telecommunications benchmark for network processors," in *Proc IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS)*, Apr 2000, pp. 154–162.

[17] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc Int Workshop on Workload Characterization (WWC-4)*, 2001, pp. 3–14.

[18] T. Gingold, "GHDL – G Hardware Design Language," 2012, disponível em ⟨http://ghdl.free.fr⟩. Acesso em 11/7/2012.