DOI: http://dx.doi.org/10.29069/forscience.2014v2n1.e50

ALBOR: UM SIMULADOR DIDÁTICO PARA AUXILIAR NO ENSINO E APRENDIZAGEM DE INSTRUÇÕES *ASSEMBLY*

Marcelo Rafael Borth[1]
Aldo Sergio de Oliveira[2]

RESUMO

Alunos iniciam e ampliam seus conhecimentos com o hardware a partir da disciplina de Arquitetura de Computadores (AC), sendo que o estudo dessa disciplina é extensivo e complexo. Com o intuito de facilitar o entendimento dos estudantes e contribuir com o corpo docente na tarefa de ministrar aulas mais atrativas utilizam-se simuladores, permitindo que processos difíceis de serem entendidos sejam vistos por meio de vários níveis de abstração. Estudar conceitos complexos a partir de simuladores acelera o entendimento e faz diferença para o aluno prosseguir seus estudos. Seguindo esse princípio, esse artigo apresenta as principais características e recursos de um simulador desenvolvido que realiza simulação da execução das instruções *assembly* do MIPS. No simulador é possível visualizar graficamente o passo a passo das instruções para a execução de um programa.

Palavras-chave: Simulador. MIPS. Arquitetura de Computadores.

A DIDATIC SIMULATOR FOR SUPPORT TEACHING AND LEARNING ASSEMBLY INSTRUCTIONS

ABSTRACT

Students start and expand their knowledge with the hardware from the discipline of Computer Architecture (CA), and the study of CA is extensive and complex. In order to facilitate the understanding of the students and to contribute to the training classes a little more easier, is used simulators, allowing processes sometimes difficult to be understood to be seen through several levels of abstraction. Studying complex concepts from a simulator accelerates the understanding and make the difference for the student to continue his studies. Following this principle, this article shows the main characteristics and features of a full-fledged simulator that performs simulation based on the implementation of MIPS assembly instructions. On the simulator it is possible view graphically a step-by-step instructions for execution of a program.

Keywords: Simulator. MIPS. Computer Architecture.

[1] Doutorando em Ciências Ambientais e Sustentabilidade Agropecuária na Universidade Católica Dom Bosco (UCDB), mestre em Ciência da Computação pela Universidade Estadual de Maringá (UEM), bacharel em Sistemas de Informação pela Universidade Paranaense e professor do Instituto Federal de Educação, Ciência e Tecnologia do Mato Grosso do Sul (IFMS) - Campus Ponta Porã. E-mail: marceloborth@gmail.com.
[2] Mestre em Ciência da Computação pela Universidade Estadual de Maringá (UEM), bacharel em Informática pela Universidade Estadual de Maringá (UEM). E-mail: aldo@analisesistemas.com.br.

1 INTRODUÇÃO

É de grande importância para os estudantes da área de informática que entendam o funcionamento da AC junto com o processador, pois o conjunto fornece um entendimento aos alunos sobre aspectos no desenvolvimento de hardware com conhecimentos fundamentais para a compreensão da lógica de programação. O uso de ferramentas visuais e didáticas pode contribuir para acelerar o aprendizado dos estudantes de AC, por meio da simulação dos conceitos e, também, tornando mais fácil o entendimento de cada interação que ocorre nas camadas de baixo-nível. Conforme afirmam Yurcyk e Gehringer (2002), a disciplina de AC é difícil para quem está ensinando quanto para quem está aprendendo. Assim, para os professores, utilizar um simulador facilita muito suas atividades e torna menos tedioso para quem está aprendendo, uma vez que normalmente é necessário preparar diversas imagens e apresentações em projetor para ilustrar o caminho dos dados em um microprocessador, o controle durante a execução, ciclo de instruções, conjunto de instruções, armazenamento de variáveis em memória, dentre outros.

Ter ferramentas gráficas que permitam ao estudante entender o conteúdo da disciplina de AC possibilita que o aprendizado seja mais compreensivo, dinâmico e interativo, e consequentemente, aumenta a capacidade de atenção do aluno, memorização, etc. Segundo Wolffe et al. (2002), vários são os motivos para os processadores terem ganhado popularidade no mundo acadêmico: i) o computador pode ser visto por meio de vários níveis de abstração; ii) os simuladores podem ser disponibilizados aos interessados muito rapidamente; iii) é encontrado simulador para diversos tópicos da área de ciência da computação e correlatas; iv) muitos livros adotam os simuladores para reforçar a didática; e, v) há muitos simuladores disponíveis gratuitamente. É diante desses objetivos que o simulador ALBOR foi proposto e desenvolvido.

O simulador ALBOR é um projeto de software que realiza a simulação da execução de instruções existentes no *Microprocessor without Interlocked Pipeline Stages* (MIPS). O foco de uso do ALBOR são os estudantes que desejam conhecer o funcionamento do processador MIPS. Este artigo tem como objetivo propor e desenvolver um software didático e interativo que simule as instruções *assembly* (linguagem de montagem) do MIPS, representando graficamente o caminho de dado e de controle. O software visa reproduzir graficamente o comportamento real de um microprocessador a partir de uma interface limpa, de fácil uso e de simples entendimento, em que as interações ocorrentes no simulador sejam absorvidas

facilmente pelo estudante. O software importa um arquivo com instruções em linguagem de montagem, convertendo em seguida o código explícito em linguagem de máquina para interpretação da simulação. Para cada instrução de máquina, há uma representação gráfica no software, mudando e mostrando o comportamento de cada interação do processador.

O artigo está organizado da seguinte maneira: a seção 2 apresenta uma descrição sobre os trabalhos relacionados à arquitetura de computadores simulando a arquitetura MIPS para processadores com fins didáticos. A seção 3 faz uma revisão bibliográfica sobre a arquitetura MIPS. A seção 4 apresenta o simulador desenvolvido neste trabalho, destacando suas principais características. E, por fim, na seção 5 são apresentadas as conclusões e direcionamentos para alguns trabalhos futuros.

2 TRABALHOS CORRELATOS

Existe vários trabalhos relacionados que demonstram o funcionamento da linguagem *assembly* no MIPS, tais como: YASS (MUSTAFA, 2013), MARS (VOLLMAR; SANDERSON, 2005), SPIM (LARUS, 2004), WebMIPS (BRANOVIC; GIORGI; MARTINELLI, 2004), J-MIPS (J-MIPS, 1997), GSPIM (BORUNDA; BREWER; ERTEN, 2006), ProcessorSim (GARTON, 2008), R10k (JUNIOR G. et al., 2007), ViSiMIPS (KABIR; BARI; HAQUE, 2011), MiniMIPS (BEM; PETELCZYC, 2003), MIPSPILOT (VEGDAHL, 2008), dentre outros. Alguns dos simuladores citados são apresentados com mais detalhes a seguir.

2.1 MARS

MARS, acrônimo para ***M****IPS* ***A****ssembler and* ***R****untime* ***S****imulator*, é um simulador para a linguagem *assembly* do MIPS desenvolvido em Java. Suas características foram embasadas especialmente para professores e estudantes (VOLLMAR; SANDERSON, 2006). Sua interface gráfica é totalmente didática, fácil de manusear e realizar as interações. Seu desenvolvimento tem como foco apresentar os dados de forma clara, para aplicação no mundo educacional (PATTERSON; HENNESSY, 2004). Esse simulador possui uma característica interessante que é a escolha de pontos de parada (*breakpoint*) para cada instrução, sendo uma opção adicional de depuração do simulador.

2.2 SPIM

Segundo Larus (2004), o SPIM é um simulador MIPS que foi muito utilizado para ministrar aulas, sendo que ele executa programas definidos pela linguagem *assembly* no MIPS (R2000/R3000), porém não utiliza o modelo de *pipeline*. O SPIM é composto por cinco janelas predominantes: janela de mensagens, segmento de texto, segmento de dados, registradores e um console que recebe entrada de dados e exibição das mensagens. Esse trabalho simula fielmente o processador MIPS, porém existem algumas diferenças de um computador convencional, tais como o *timing* das instruções e o sistema da memória que não são idênticos. Esse projeto também não simula *cache* de memória e não reflete corretamente o retardamento de operações de ponto flutuante, multiplicação ou divisão.

2.3 WebMIPS

O WebMIPS é um ambiente de simulação do MIPS desenvolvido em Java que pode ser utilizado por vários usuários ao mesmo tempo. Sua principal vantagem é ser acessível pela Internet, não sendo necessário instalar qualquer programa ou extensão para acessá-lo. Sua interface é interessante, pois representa de forma facilitada os componentes e barramentos de um processador MIPS. Seu principal objetivo é a criação da simulação dos cinco estágios passo a passo do funcionamento do *pipeline*, com a entrada e saída de dados de todos os seus elementos, executando instruções previamente definidas.

2.4 J-MIPS

O J-MIPS é um simulador MIPS desenvolvido em Java. Sua execução é pela Internet e o programa é executado a partir de *applets*. Por ser uma aplicação Web, ele não possui uma interface gráfica limitada, pois é possível visualizar a animação do controle de dados pelo *applet*.

3 ARQUITETURA MIPS

A implementação do MIPS é baseada na arquitetura RISC, que possui um conjunto reduzido de instruções, *clock* único, código grande e dá ênfase no software. A origem da

arquitetura MIPS deu-se em 1981 a partir de um projeto de pesquisa na Universidade de Stanford, tendo sido difundido posteriormente pela MIPS Computer Systems. Seu principal desenvolvimento foi direcionado a microprocessadores embutidos para produtos eletrônicos.

A arquitetura MIPS é estudada em muitas universidades e centros educacionais, facilitando a compreensão dos conceitos abstratos de *design* de computação (BRANOVIC; GIORGI; MARTINELLI, 2004). Outra vantagem é de ser utilizada por professores em diversas universidades do mundo como referência para o ensino de AC (BRANOVIC; GIORGI; MARTINELLI, 2004), além de ser adotado em livros didáticos (PATTERSON; HENNESSY, 2004). O fato de estar relacionada a grande utilização para fins educacionais é consequência da arquitetura ser bem desenhada e "limpa" (VOLLMAR; SANDERSON, 2006) que, posteriormente, até influenciou o desenvolvimento de processadores como o SPARC.

O MIPS é um processador que usa a linguagem de montagem *assembly* para sua execução. Essa linguagem é uma representação baseada em símbolos de uma linguagem binária de máquina. Pelo fato de interpretar os símbolos e executá-los em uma linguagem de máquina, facilita e torna mais adequada a implementação pelos seres humanos. O MIPS pode ser classificado na família dos processadores de 32-bits e 64-bits e são usados por muitos sistemas embarcados, tais como: impressoras, PDAs, roteadores e videogames como o Playstation® (PINCKNEY et al., 2008). Por outro lado, na área de computadores de mesa pessoais não obteve sucesso, pois outros processadores como a Intel® predominaram no mercado.

Conforme apresentado por MIPS Technologies (2001), as arquiteturas MIPS32 e MIPS64 fornecem vantagem quanto ao custo e a performance se comparadas a outras implementações dos processadores que se baseiam em projetos de arquitetura tradicional. Logo, essa é uma vantagem adquirida pelo MIPS por meio da organização, processos da tecnologia e estrutura do compilador.

Evoluir uma arquitetura é viável a partir do momento em que o produto está atendendo a um vasto mercado, propiciando vários benefícios e mantendo uma arquitetura escalar. O MIPS possui uma comunidade que avalia frequentemente sugestões de mudanças e melhorias na arquitetura. Essas mudanças são necessárias para fornecer uma plataforma estável, bem como atender a demanda de novos mercados. Assim, a arquitetura MIPS evolui mantendo o compromisso de envolver os recursos de software e hardware, sendo que o desenvolvimento da arquitetura MIPS64 é totalmente compatível com a arquitetura MIPS32, diferente do que

acontece com alguns Sistemas Operacionais atualmente. Isso significa que os programas criados podem ser executados em qualquer processador MIPS (MIPS TECHNOLOGIES, 2001).

A arquitetura MIPS está desacoplada da implementação de hardware, deixando a estrutura do microprocessador livre para criar hardwares proprietários dentro da definição arquitetural. Isso acontece porque a arquitetura se diferencia da implementação de hardware. A arquitetura refere-se ao conjunto de instruções, registradores, modelo de exceções, gerenciamento de memória e outras características que todos os hardwares executam. A implementação, por sua vez, refere-se ao caminho específico do processador aplicado na arquitetura. Além disso, possui um conjunto de instruções reduzidas, permitindo, assim, maior facilidade de compreensão, entendimento e aprendizado (VOLLMAR; SANDERSON, 2006). Essa arquitetura foi preparada para simplificar todo o conjunto de instruções determinadas, sem prejudicar sua flexibilidade. Esse projeto foi criado baseando-se em escalabilidade, por isso, instruções em processadores mais poderosos estão ausentes nessa arquitetura. Entretanto, essa limitação é suprida por meio da utilização de pseudo-instruções disponibilizadas para utilização pelos programadores mediante linguagem de montagem.

O processador possui um total de 32 registradores de uso geral de 32 bits cada um. Em decorrência do número de registradores existente no MIPS, deve ser considerado que os programadores da linguagem *assembly* precisam preocupar-se em guardar o estado dos registradores em memória, pois se torna necessário garantir que os dados existentes não sejam perdidos. A Tabela 1 ilustra a convenção dos registradores utilizados na arquitetura MIPS (SWEETMAN, 2006).

Uma característica relevante da arquitetura MIPS é a realização de operações *load-store* na memória, ou seja, somente as instruções de carregar (*load*) e armazenar (*store*) possuem acesso a memória. Portanto, o processador disponibiliza um conjunto de registradores para acesso aos dados, isso tende a reduzir o número de acesso à memória, visto que, esse acesso é bem mais custoso em relação ao acesso aos registradores do processador. Baseando-se no funcionamento da memória da arquitetura MIPS é possível chegar até ela apenas por meio de instruções de carregamento e armazenamento.

Na arquitetura MIPS a memória possui três divisões importantes: o segmento de texto, o segmento de dados e o segmento de pilha, conforme ilustrado na Figura 1. O segmento de texto é o local onde são armazenadas as instruções a serem executadas pelo programa, sendo que essa parte da memória possui tamanho estático e não pode ser expandido. O segmento de

dados, localizado acima do segmento de texto, tem por função o armazenamento de valores conforme a execução do programa. Esse segmento é subdividido em duas categorias: dados estáticos e dados dinâmicos. Os dados estáticos armazenam valores durante toda a execução do programa como, por exemplo, as variáveis globais de uma linguagem de programação que são alocadas estaticamente e podem ser acessadas a qualquer momento durante a execução do programa. Por outro lado, os dados dinâmicos armazenam valores que são criados durante a execução do programa. Esses dados podem não durar por todo o ciclo de vida da aplicação e seu tamanho também pode ser expandido conforme sua necessidade. Como não há possibilidade de descobrir quantas informações serão armazenadas no segmento de dados dinâmico e no segmento de pilha, o Sistema Operacional se responsabiliza de gerenciar esses crescimentos, redimensionando a memória nesses segmentos quando necessitar, podendo aumentar o suficiente para ocupar todo o espaço da memória. Para permitir a expansão, os dois segmentos que podem ser expandidos dinamicamente estão bem afastados um do outro e podem utilizar o limite da memória disponível no endereçamento, conforme ilustra a Figura 1.

Tabela 1 - Convenção de uso dos registradores do MIPS

Nome	Número	Convenção de Utilização
$zero	0	Armazena a constante 0 (zero)
$at	1	Reservado para o *assembly* (*assembly temporary*)
$v0, $v1	2-3	Usado para avaliar expressões e guardar o resultado da função
$a0 - $a3	4-7	Argumento de funções (*argument*)
$t0 - $t7	8-15	Valor temporário (*temporary*)
$s0 - $s7	16-23	Registro de variáveis para sub-rotinas (*subroutine*)
$t8, $t9	24, 25	Valor temporário (*temporary*)
$k0, $k1	26, 27	Reservado para o *kernel* do Sistema Operacional
$gp	28	Ponteiro para a área global (*global pointer*)
$sp	29	Apontador de pilha (*stack pointer*)
$fp	30	Apontador de quadro (*frame pointer*)
$ra	31	Registrador de endereço de retorno (*return address*)

Fonte: Sweetman (2006).

O endereçamento de memória no MIPS é baseado em byte, logo cada endereço de memória possui 1 byte (8 bits) para armazenamento. Quando os dados e endereços na arquitetura MIPS são de 32 bits podemos dizer que a palavra do processador é de 32 bits.

Todas as instruções ocupam o mesmo espaço, 1 palavra em memória, ou seja, 32 bits que podem ser alocadas em posições consecutivas na memória.

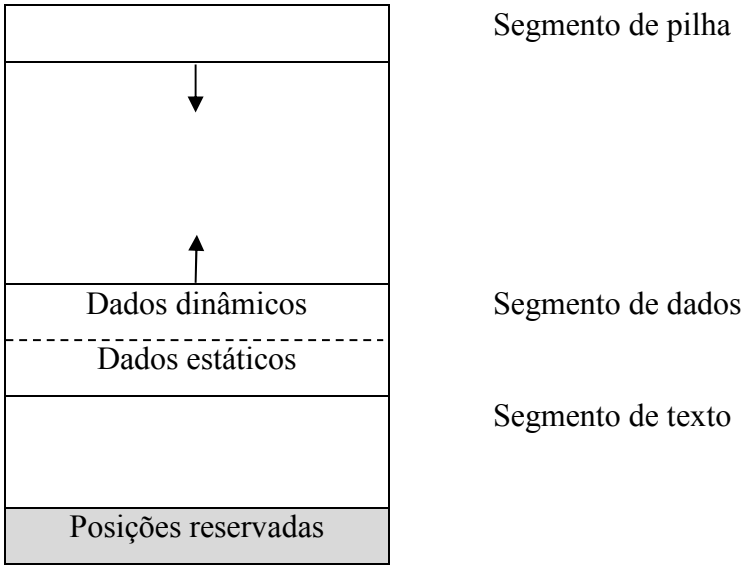

Figura 1 - Layout da Memória
Fonte: Adaptado de Larus (2004).

Para suas instruções, o MIPS fornece um formato de especificação para a linguagem *assembly*, facilitando o desenvolvimento dos programas. Existem alguns formatos de instruções que merecem ser citados como, instruções lógicas, instruções aritméticas, instruções de comparação, instruções de desvio, instruções de carga e instruções de movimentação. As instruções lógicas e aritméticas possuem funções que realizam comparação e soma do conteúdo dos registradores. Instruções de comparação têm como função comparar dois registradores e armazenar o valor resultante em um terceiro registrador. Instruções de carga são responsáveis por carregar os valores da memória a um registrador, ou vice-versa. Instruções de movimentação de dados são responsáveis por modificar o conteúdo entre registradores. As instruções criadas para essa arquitetura são executadas em até cinco estágios (PINCKNEY et al., 2008):

- 1º Ciclo – busca da instrução (*fetch*);
- 2º Ciclo – decodificação (*decode*);
- 3º Ciclo – execução (*execute*);
- 4º Ciclo – acesso a memória (*memory*); e,
- 5º Ciclo – atualização dos registradores (*write-back*).

Pinckney et al. (2008) define cada estágio ocorrido para a execução das instruções. Normalmente, cada instrução ao terminar sua execução executa os cinco ciclos citados, porém em algumas das instruções, algumas fases podem não ser executadas. As instruções são lidas a partir da *cache* durante o primeiro ciclo, a partir do endereço armazenado em memória. Durante o estágio de decodificação os dados são lidos a partir de um arquivo de registro. No ciclo de execução são realizadas operações aritméticas e desvio de valores. Leituras e escritas são realizados durante o estágio de memória. Por fim, o último ciclo realiza a gravação dos valores no arquivo de registro.

Na arquitetura do MIPS não existe qualificadores sobre execução de operação, ou seja, não existe registradores para armazenar informações que geram um maior armazenamento de capacidade quando ocorre *overflow*. A identificação de situações como essa deve ser realizada via software, porém para verificar essas exceções pode ser utilizada instruções da arquitetura.

4 SIMULADOR PROPOSTO

Para iniciar a construção do simulador ALBOR foi necessário um estudo prévio sobre o funcionamento do microprocessador MIPS, entender o que é uma linguagem de máquina, compreender instruções de máquina e qual o seu comportamento em um processador, uma vez que o software faz uma representação gráfica de cada instrução. Além disso, foi estudado rapidamente a Interface de Desenvolvimento Integrado (IDE) e a linguagem de programação utilizada no desenvolvimento, uma vez que os autores já tinham conhecimento das tecnologias adotadas.

O presente simulador foi desenvolvido utilizando o Delphi 7.0. Sua escolha é sustentada por ser uma ferramenta prática e de fácil uso, que possui uma vasta bibliografia para auxílio. A linguagem de programação utilizada foi o *Object Pascal*. Ambas possuem um bom desempenho, o que justifica sua larga adoção por empresas e profissionais da área de desenvolvimento de software.

O simulador ALBOR oferece uma grande quantidade de informação de forma simples e objetiva e, consequentemente, contribui ao processo de aprendizagem do estudante de AC. O simulador desenvolvido faz a análise das instruções informadas pelo usuário, criando, assim, uma exibição gráfica para o caminho de dados e de controle das instruções do MIPS monociclo.

Para cada instrução executada pelo simulador haverá um ponteiro em destaque no segmento de texto que informará a instrução atual, isso irá ocorrer conforme for prosseguindo sua execução, até o termino de todas as instruções. A mesma situação ocorre para os 32 registradores, pois quando receber uma instrução para gravar um valor em um registrador, haverá um apontador (linha em destaque) que mostrará qual o registrador está sendo alterado. Por exemplo, a instrução "sw" (*store word*) que faz o armazenamento de dado irá primeiramente acessar a memória de dados, assim, a posição que irá ser acessada aparecerá em destaque, lembrando que o simulador destaca a próxima instrução e não a que está sendo executada no momento, da mesma forma como acontece com os métodos de depuração.

4.1 Conteúdo abordado pelo simulador ALBOR

A seguir são apresentados os conteúdos abordados pelo simulador ALBOR:

1. Espaço reservado para edição das instruções (ver Figura 2);
2. Menu principal (ver Figura 3);
3. Barra de ferramentas (ver Figura 3);
4. Segmento de texto (local onde ficam carregadas as instruções a ser executadas pelo simulador) (ver Figura 3);
5. Tabela de registradores (ver Figura 3);
6. Segmento de dados (memória ou pilha) (ver Figura 3); e,
7. Opções complementares (ver Figura 3).

O item 1 mostra o primeiro passo a ser feito para prosseguir a simulação, que é a criação das instruções a serem executadas pelo simulador. O usuário possui duas opções para gerar as instruções: criar o programa com as instruções diretamente na aba "Texto fonte", em que cada instrução deverá ficar em uma linha, ou criar um arquivo (extensão ".asm") com todas as instruções do programa que deseje executar dentro do arquivo, e carregá-lo.

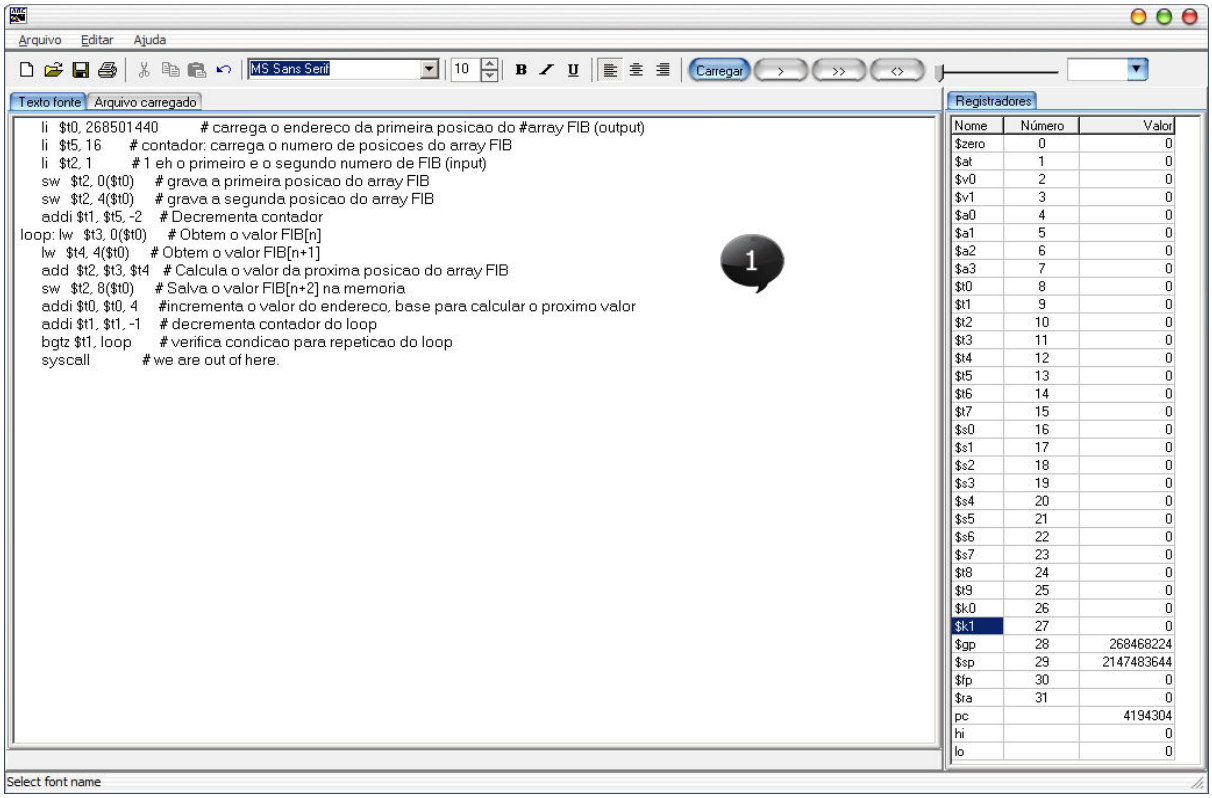

Figura 2 - Janela de edição de instrução do simulador ALBOR
Fonte: Desenvolvido pelos autores - resultado deste trabalho.

O item 2 possui um menu com opção de salvar o programa com as instruções, um menu de ajuda, dentre outros. O item 3 apresenta possui as opções de mudar o tema (*skin*) do ALBOR, carregar o arquivo de instruções, executar o programa, depurar o programa passo a passo, parar a execução do programa, dentre outros. O segmento de texto, mostrado no item 3, é o local onde carregam-se as instruções a serem executadas, sendo que cada instrução tem um endereço de memória pré-determinado, podendo ser acompanhado na execução por meio do endereço que estiver no *program counter* (PC). No item 5 são mostrados os registradores, que tem como objetivo reduzir o número de acesso à memória, uma vez que é muito mais rápido acessar um registrador em vez de acessar a memória externa para recuperar um dado armazenado. No item 6 é ilustrada a memória distribuída em duas divisões, segmento de dados e segmento de pilha, ambos podem expandir-se conforme suas necessidades durante o ciclo de vida de um programa. No item 7, foram adicionadas algumas opções para monitorar os endereçamentos em hexadecimal. Assim, ao permitir visualização em hexadecimal, a mudança ocorre tanto para o segmento de dados quanto para o segmento de texto.

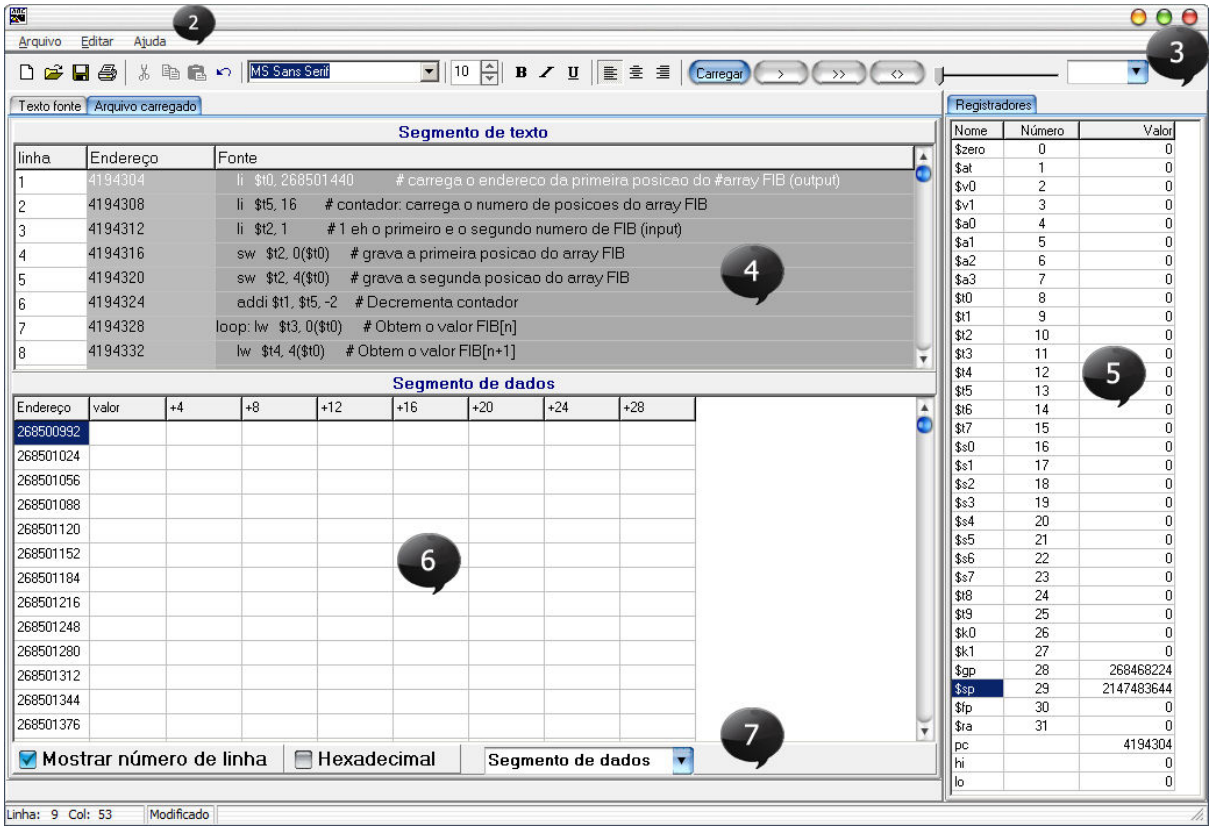

Figura 3 - Janela principal do simulador ALBOR
Fonte: Desenvolvido pelos autores - resultado deste trabalho.

4.2 Instruções implementadas pelo simulador ALBOR

O projeto ALBOR implementa algumas instruções, tais como: instruções lógicas, instruções aritméticas, instruções de comparação, instruções de desvio, instruções de carga e instruções de movimentação. A seguir são apresentadas as instruções desenvolvidas no simulador (o Anexo 1 ilustra um exemplo de um programa para a execução do simulador proposto):

- **Instruções lógicas**: AND, OR e XOR;
- **Instruções aritméticas**: ADD, ADDI, ADDU, ADDIU, SUB, DIV, MULT, MFHI, MFLO, MTLO, MTHI e LI;
- **Instruções de desvios**: BEQ, BGEZ, BGTZ, BLEZ, BLTZ e BNE;
- **Instruções de deslocamento**: SLL, SRL e MOVE;
- **Transferência de dados**: SW e LW;
- **Desvio incondicional**: JAL, JALR, JR e J; e,
- **Outras instruções**: NOP e SYSCALL.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi proposto e desenvolvido um simulador MIPS para auxiliar no ensino e aprendizagem da disciplina de arquitetura de computadores. Houve grande cuidado no desenvolvimento do simulador para que ele ficasse de fácil utilização, didático, dinâmico e interativo para o usuário. O simulador apresenta uma interface simplificada e dinâmica, em que o usuário tem a opção de ver exatamente o que está ocorrendo a cada instrução executada.

Como trabalhos futuros, o simulador proposto neste trabalho será implementado na linguagem Java para viabilizar o uso em múltiplos Sistemas Operacionais. Além disso, em sua nova versão serão desenvolvidos recursos do *pipeline*, bem como o restante das instruções não implementadas do MIPS, permitindo assim um conjunto mais amplo para realização das simulações.

REFERÊNCIAS

BEM, E. Z.; PETELCZYC, L. MiniMIPS: a simulation project for the computer architecture laboratory. In: TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION (SIGCSE '03), 34., 2003, NY. **Proceedings...** NY, USA: ACM Press, 2003. p. 64-68.

BORUNDA, P.; BREWER, C.; ERTEN, C. GSPIM: graphical visualization tool for MIPS Assembly programming and simulation. In: TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION (SIGCSE), 37., 2006, NY. **Proceedings...** NY: [s.n.], 2006. p. 244-248.

BRANOVIC, I.; GIORGI, R.; MARTINELLI, E. WebMIPS: a new web-based MIPS simulation environment for computer architecture education. In: WORKSHOP ON COMPUTER ARCHITECTURE EDUCATION: HELD IN CONJUNCTION WITH THE INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 31ST., 2004, NY. **Proceedings...** NY: ACM, 2004. p. 93-98.

GARTON, J. **ProcessorSim MIPS:** a visual MIPS R2000 processor simulator, 2008. Disponível em: <http://jamesgart.com/procsim/>. Acesso em: 11 dez. 2013.

J-MIPS (The Java MIPS simulator**)**, 1997. Disponível em: <http://www.csse.monash.edu.au/packages/jmips/>. Acesso em: 02 set. 2013.

JUNIOR G., N. A. et al. R10k: um simulador de arquitetura superescalar. In: WORKSHOP SOBRE EDUCAÇÃO EM ARQUITETURA DE COMPUTADORES (WEAC), 2007, Gramado. **Anais...** Gramado: UFRGS/SBC, 2007. p. 23-31.

KABIR, M. T.; BARI, M. T.; HAQUE, A. L. ViSiMIPS: visual simulator of MIPS32 pipelined processor. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE & EDUCATION (ICCSE), 6th., 2011, Vancouver. **Proceedings...** Vancouver: IEEE, 2011.

LARUS, J. **SPIM S20**: a MIPS R2000 Simulator. Madinson: Technical Report, Computer Sciences Department, University of Wisconsin, 2004. Disponível em: <http://phoenix.goucher.edu/~kelliher/f2005/cs220/spim.pdf>. Acesso em: 02 set. 2013.

MIPS TECHNOLOGIES, INC. **MIPS32 architecture for programmers**: introduction to the MIPS32 architecture. [S.l.: s.n.], 2001. v. 1.

MUSTAFA, B. YASS: a system simulator for operating system and computer architecture teaching and learning. **European Journal of Science and Mathematics Education**, v. 1, n. 1, p. 34-42, 2013.

PATTERSON, D.; HENNESSY, J. L. **Computer organization and design:** the hardware/software interface. 3. ed. San Francisco, CA: Morgan Kaufmann, 2004.

PINCKNEY, N. et al. A MIPS R2000 Implementation. In: ANNUAL DESIGN AUTOMATION CONFERENCE (DAC´08), 45., 2008, New York. **Proceedings...** New York: [s.n.], 2008. p. 102-107.

SWEETMAN, D. **See MIPS Run**. 2. ed. [S.l.]: Morgan Kaufmann, 2006.

VEGDAHL, S. R. MIPSPILOT: a compiler-oriented MIPS simulator. **Journal of Computing Sciences in Colleges**, v. 24, n. 2, p. 32-39, 2008.

VOLLMAR, K.; SANDERSON, P. MARS: an education-oriented MIPS Assembly language simulator. In: TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION (SIGCSE '06), 37th., 2006, New York. **Proceedings...** New York: ACM, 2006. v. 38, p 239-243.

_____. MIPS Assembly language simulator designed for education. **The Journal of computing Sciences in Colleges**, v. 21, n. 1, 2005.

WOLFFE, G. S. et al. Teaching computer organization/architecture with limited resources using simulators. In: TECHNICAL SYMPOSIUM OF COMPUTER SCIENCE EDUCATION, 33rd., 2002. [S.l.]. **Proceedings…** [S.l.]: ACM Press, 2002. v. 34, n. 1, p. 176-180.

YURCYK, W.; GEHRINGER, E. F. A survey of web resources for teaching computer architecture. In: WORKSHOP ON COMPUTER ARCHITECTURE EDUCATION (WCAE), 2002, Anchorage. **Proceedings...** Anchorage: [s.n.], 2002. p. 126-131. Disponível em: <http://www.ncsu.edu/wcae/ISCA2002/submissions/yurcik.pdf>. Acesso em: 02 set. 2013.

ANEXO 1 – Programa exemplo para execução no ALBOR

Esse anexo apresenta um programa exemplo criado em linguagem de montagem para executar no simulador ALBOR. O exemplo calcula os primeiros 18 números de Fibonacci e salva um vetor na memória. O exemplo contém uma instância para cada instrução do simulador proposto, conforme apresentado a seguir na Quadro 1.

```
main:
      li   $s0, 268501440
      li   $s1, 18
      li   $s2, 1
      sw   $s2, 0($s0)
      sw   $s2, 4($s0)
      addi $s1,$s1,-2

 loop:  addi $sp,$sp,-8
      sw   $s0,4($sp)
      sw   $s1,0($sp)
      jal  obtemInput
      move $s0,$a0
      move $s1,$a1
      jal geraOutput
      lw $s1,0($sp)
      addi $sp,$sp,4
      lw $s0,0($sp)
      addi $sp,$sp,4
      jal contadorLoop
      bgtz $s1, loop
      syscall

obtemInput:
 nop
     lw $a0,0($s0)
     lw $a1,4($s0)
     jr $ra
     nop

 geraOutput:
     add $a2,$s0,$s1
     addi $sp,$sp,-4
     sw $ra,0($sp)
     jal salvaOutput
     lw $ra,0($sp)
     addi $sp,$sp,4
     jr $ra

salvaOutput:
     lw $s0,8($sp)
     sw $a2,8($s0)
     jr $ra
```

```
contadorLoop:
    addi $s0,$s0,4
    addi $s1,$s1,-1
    jr $ra
    nop
```

Quadro 1 - Programa exemplo para executar no simulador ALBOR
Fonte: Programa exemplo resultado deste trabalho.

Recebido em: 16/09/2013

Aprovado em: 01/02/2014