SimuS – Um Simulador Para o Ensino de Arquitetura de Computadores

Conference Paper · October 2016

CITATIONS
7

READS
2,359

2 authors:

Gabriel P. Silva
Federal University of Rio de Janeiro
75 PUBLICATIONS **53** CITATIONS
SEE PROFILE

José Antonio Dos Santos Borges
Federal University of Rio de Janeiro
57 PUBLICATIONS **142** CITATIONS
SEE PROFILE

SimuS

Um Simulador Para o Ensino de Arquitetura de Computadores

Gabriel P. Silva
Departamento de Ciência da Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
gabriel@dcc.ufrj.br

José Antônio dos S. Borges
Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
antonio2@nce.ufrj.br

***Abstract* — Este artigo apresenta o simulador SimuS, desenvolvido para uma arquitetura do processador hipotético Sapiens, especialmente concebido para o ensino de arquitetura de computadores, a partir da experiência adquirida no uso e desenvolvimento do simulador Neanderwin para o processador Neander-X. No processador hipotético Sapiens, a arquitetura e o conjunto de instruções inicialmente propostos para o processador Neander-X são estendidos, de forma a obtermos uma arquitetura de processador com mais recursos, permitindo o seu uso desde cursos iniciais até cursos mais avançados.**

Keywords—ensino; arquitetura de computadores; Neander; Neanderwin; Sapiens; Simus; simulador

I. INTRODUÇÃO

Um dos problemas encontrados no ensino de arquitetura de computadores é fazer com que os alunos compreendam corretamente o funcionamento de um processador, proporcionando também uma visão comparativa sobre algumas possibilidades arquiteturais. O uso de simuladores de processadores, sejam eles comerciais ou puramente didáticos, é uma estratégia comumente utilizada para alcançar estes objetivos. A visão dos componentes, do funcionamento da arquitetura do processador, do formato das instruções e dos passos necessários para a execução de um programa, são elementos importantes na formação de um conhecimento sólido do funcionamento dos processadores por parte do estudante.

Ao longo de mais de 10 anos, o simulador Neanderwin [1] e o processador Neander-X têm sido utilizados, pelos autores e por diversos professores em universidades do país, nos cursos iniciais da disciplina de Arquitetura de Computadores. A partir da experiência adquirida, iniciamos o desenvolvimento do simulador SimuS, para o processador hipotético Sapiens, que são ambos apresentados neste artigo.

A filosofia principal do simulador SimuS é apresentar uma arquitetura que possa ser explorada em diversos níveis de complexidade, com o emprego de exemplos mais simples, para os cursos iniciais, até exemplos mais complexos, para os cursos mais avançados, sempre com o uso da mesma ferramenta. Pretendemos ainda que essa ferramenta tenha um uso ampliado, tanto para os cursos de Ciência da Computação, como Engenharia de Computação ou Sistemas de Informação.

A arquitetura e o conjunto de instruções inicialmente propostos para o processador Neander-X são estendidos no processador Sapiens, de modo a obtermos uma arquitetura de processador com maiores possibilidades. Para isso foram introduzidas mudanças que incluem um maior espaço de endereçamento de memória, a chamada e retorno de procedimentos, um apontador de pilha e um conjunto mais robusto de instruções, além da emulação de novos dispositivos de E/S e atualização da interface de usuário, para representar adequadamente essas novas modificações. Apesar dessas mudanças, não houve perda de compatibilidade, em nível de linguagem de montagem, com códigos produzidos para as arquiteturas legadas do Neander e Neander-X, ainda podem ser compilados e executados no simulador SimuS.

O simulador SimuS apresenta um ambiente integrado de desenvolvimento, onde o aluno pode editar, compilar, depurar e executar código de programas escritos na linguagem de montagem do processador Sapiens. Na elaboração deste simulador nos mantivemos fiéis aos conceitos que apresentamos anteriormente, de uso de um ambiente integrado de desenvolvimento, envolvendo desde a inclusão de um editor do código com uma ferramenta de apoio ao desenvolvimento inicial do código, passando por um montador com informações sobre os erros de sintaxe e finalmente um simulador, onde o funcionamento final do programa pode ser verificado, com acesso a todos os elementos que compõem o estado do processador como acumulador, apontador de instruções, códigos de condição, memória, entre outros. O simulador SimuS, por ser distribuído em código aberto [1], viabiliza a sua expansão (por outros professores ou por alunos em projeto), possibilitando a exploração de variantes da arquitetura ou adição de novas ferramentas de ensino ou projeto. O código fonte deste simulador pode ser compilado para obtermos versões binárias para execução tanto para o sistema operacional Windows, como para o sistema operacional Linux e suas variantes.

Este artigo está organizado da seguinte maneira: na Seção II apresentamos os trabalhos correlatos, na Seção III a arquitetura do processador didático Sapiens é detalhada e apresentamos suas instruções e um exemplo de programa em linguagem de montagem. Na seção seguinte discutimos a interface de usuário do SimuS e finalmente apresentamos as conclusões e trabalhos futuros.

[1]Disponível em https://sourceforge.net/projects/simus/

II. Trabalhos Correlatos

Existem muitas ferramentas de simulação para processadores e microcontroladores comerciais. Os simuladores comerciais mais usados em projetos de equipamentos na atualidade (e que em princípio poderiam ser usados nos cursos básicos de arquitetura de computadores) são os de 8051, ATmega, PIC e variantes de processadores do tipo RISC. São sistemas completos, contendo muitas ferramentas integradas, mas quase todos, sendo destinados ao uso profissional, tendo alguns deles licenças de uso pagas e de uso relativamente complexo, inviabilizando o uso amplo no ensino de graduação em muitos cursos no Brasil.

Na área dos simuladores didáticos existem vários sistemas desenvolvidos no exterior, muitos deles descritos em [3]. Como exemplos de simuladores didáticos de processadores comerciais podemos citar o Abacus [9], o GNUSim8085 [4], MARS [11] e o WinMIPS64 [5]. O Abacus (desenvolvido para Windows) e o GNUSim8085 (desenvolvido para plataformas Unix) simulam o microprocessador 8085 da Intel, que tem sido utilizado amplamente no ensino de arquitetura de computadores, devido à sua simplicidade e por não terem pipeline. Ambos apresentam os valores dos registradores no decorrer da execução das instruções, além de exibir o conteúdo presente na memória do processador. A arquitetura do processador MIPS, que utiliza o pipeline, tem disponíveis os simuladores MARS (32 bits) e WinMIPS64 (64 bits). Essas ferramentas possibilitam a visualização das instruções passando pelos diversos estágios do pipeline, além de examinar o conteúdo de todos os registradores, sejam inteiros ou de ponto flutuante, e também da memória, convenientemente dividida entre memória de dados e de instruções.

Há alguns exemplos de ferramentas didáticas desenvolvidas para processadores hipotéticos, entre as quais podemos citar o simulador R2DSim [6], uma ferramenta didática para simulação de uma arquitetura RISC de 16 bits com pipeline de quatro estágios, cujo banco de registradores possui tamanho e largura configurável. No conjunto de ferramentas apresentadas estão um montador e um simulador integrados. O SIMAEAC [7] - Simulador Acadêmico para Ensino de Arquitetura de Computadores - implementa um subconjunto da arquitetura do processador 8085, mas com uma memória de apenas 256 bytes, com um montador e simulador integrados. Esses dois primeiros exemplos não possuem editor integrado e não é claro como eventuais mensagens de erro são apresentadas. Temos ainda o SEAC [8], um simulador online para uma arquitetura hipotética com pipeline de quatro estágios, com foco nos passos de microprogramação necessários para executar cada instrução. Outro simulador interessante é o W-Neander, produzido como software companheiro do livro Fundamentos de Arquitetura de Computadores [2], mas que carece de montador e de editor integrados. A arquitetura Neander é muito simples e pode ser explicada em uma ou duas aulas. Avaliamos que o conjunto original de instruções do Neander apresenta algumas limitações que dificultam a criação de programas pouco mais do que triviais. Embora na obra desses autores sejam apresentadas outras arquiteturas mais sofisticadas, porem há necessidade de migração de uma ferramenta para outra, na medida em que a complexidade dos processadores aumenta.

Neste sentido, apresentamos em 2006 o simulador Neanderwin para uma versão estendida dessa arquitetura, que chamamos de Neander-X, ambos descritos em detalhes em [1]. Nossa ideia foi unir as estratégias de integração de ferramentas, encontradas nas ferramentas profissionais com uma arquitetura simples como a do Neander, que seria expandida para diminuir algumas limitações e ampliar o seu potencial didático.

Consolidamos e avançamos nesses conceitos, apresentando aqui tanto o simulador SimuS quanto o processador Sapiens, que são uma evolução do simulador Neanderwin e da arquitetura Neander-X, mas sempre guardando compatibilidade em nível de linguagem de montagem com os programas já desenvolvidos para os processadores originais.

III. A Arquitetura do Processador Sapiens

A. *Apresentação*

A arquitetura do processador Sapiens pode ser facilmente associada a uma evolução do processador Neander-X que, com sua simplicidade, é suficiente para introduzir de forma suave os aspectos centrais do funcionamento de um processador e de sua programação para resolver problemas simples. Ao longo dos anos, porém, observamos diversas limitações quando problemas um pouco mais "reais" eram apresentados aos alunos, em particular:

- Quantidade muito limitada de memória (256 bytes) que restringia o tamanho do programa ou dos dados processados;
- Grande dificuldade de realizar chamadas de rotinas pela ausência de instruções e mecanismos convenientes;
- Ausência de *flags* para suporte a aritmética multi-byte;
- Limitação do modo de endereçamentos, que dificultava até a manipulação de estruturas de dados muito simples.

Desta forma, problemas interessantes acabavam sendo deixados de lado, seja pela impossibilidade ou pelo tempo necessário para desenvolvê-los.

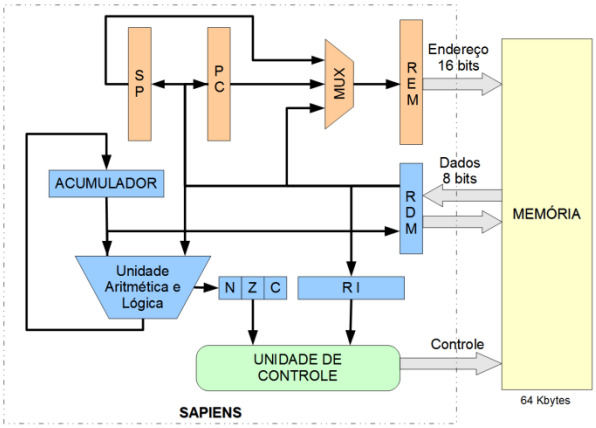

Fig. 1. Diagrama em Blocos do Processador Sapiens

A arquitetura do processador Sapiens, cujo diagrama em blocos é mostrado na Figura 1, incluiu, entre outros detalhes:

- Um modo indireto de endereçamento, possibilitando exercitar as noções de indexação e ponteiros – que são fundamentais para entendimento de qualquer estrutura básica de programação;
- O modo imediato de acesso aos operandos, simplificando as operações de atribuição de dados;
- Operações de entrada e saída de dados para dispositivos E/S, em espaço de endereçamento separado da memória;
- Incremento da largura do apontador de instruções (PC) para 16 bits, permitindo endereçar até 64 Kbytes.
- Um apontador de pilha (SP, do inglês *stack pointer*), também de 16 bits, para possibilitar a chamada e o retorno de rotinas e procedimentos;
- Um código de condição (*flag*) C (do inglês *carry)* para o vai-um e também vem-um;
- Uma instrução de TRAP para chamada do simulador para realizar operações mais elaboradas de E/S.
- Um conjunto novo de instruções de movimentação de pilha, deslocamento do registrador, soma e subtração com vai-um/vem, entre outras.

B. Formato das Instruções e Modos de Endereçamento

As instruções em linguagem de máquina do processador Sapiens podem ter um, dois ou três bytes (8 bits), conforme pode ser visto na Figura 2. Nas instruções, o primeiro byte sempre contém o código de operação nos 6 bits mais significativos e, quando for o caso, o modo de endereçamento nos 2 bits menos significativos. As instruções com apenas um byte não tem outro operando além do acumulador, que é um operando implícito para quase todas as instruções. As instruções com dois bytes são aquelas que, além do acumulador, usam um dado imediato de 8 bits como operando no segundo byte da instrução.

Fig. 2. Formato da Instrução do Processador Sapiens

Nas instruções com 3 bytes os dois últimos bytes podem conter o endereço de memória do operando (modo direto), ou o endereço de memória do ponteiro para o operando (modo indireto) ou ainda o próprio operando (modo imediato de 16 bits). Note que nos dois primeiros modos de endereçamento o operando em si possui apenas um byte de comprimento. A codificação para o modo de endereçamento é a seguinte:

00 - Direto: o segundo e terceiro bytes da instrução contêm o endereço do operando;

01 - Indireto: o segundo e terceiro bytes da instrução contêm o endereço da posição de memória com o endereço do operando (ou seja, é o endereço do ponteiro para o operando). Na linguagem de montagem, usou-se como convenção para indicar que um operando é indireto precedê-lo pela letra "@" (arroba);

10 - Imediato 8 bits: o segundo byte da instrução é o próprio operando. Na linguagem de montagem, usou-se como convenção para indicar que um operando é indireto precedê-lo pela letra "#" (tralha).

11 - Imediato 16 bits: os dois bytes seguintes à instrução são utilizados como operando. Na linguagem de montagem, usou-se como convenção para indicar que um operando é indireto precedê-lo pela letra "#" (tralha). O montador fica encarregado de gerar a codificação do operando no tamanho correto de acordo com a instrução. A única instrução que utiliza este modo é a LDS (Load Stack Pointer).

C. Códigos de Condição

A seguir são apresentados os códigos de condição do processador Sapiens, ou seja, *flags* que indicam o resultado da última operação realizada pela UAL.

- **N – (negativo):** 1 – o resultado é negativo; 0 – o resultado não é negativo
- **Z – (zero):** 1 – indica que o resultado é igual a zero; 0 – indica que o resultado é diferente de zero
- **C – (vai-um):** 1 – indica que a última operação resultou em ”vai-um” (*carry*), no caso de soma, ou ”vem-um” (*borrow*) em caso de subtração; 0 – o resultado não deu nem ”vai-um” ou ”vem-um”.

D. Descrição das Instruções

O conjunto original de instruções foi expandido para permitir uma maior capacidade de processamento. Todas as instruções originais do Neander-X foram mantidas, com exceção da instrução LDI (*load immediate*), que ainda é aceita pelo novo montador, mas na geração do programa em linguagem de máquina foi substituída pela instrução LDA #imed, que realiza a mesma função. Ou seja, um código escrito para os processadores Neander ou Neander-X é totalmente compatível e executado sem problemas no SimuS.

Como destaque para as novas instruções introduzidas na arquitetura temos a adição e subtração com carry, (ADC e SBC), novas instruções lógicas como “ou” exclusivo, deslocamento para a direita e esquerda do acumulador (XOR, SHL, SHR e SRA), novas instruções de desvio condicional (JP, JC, JNC), instruções para chamada e retorno de procedimento (JSR e RET), instruções para a manipulação da pilha (PUSH e POP), além da própria inclusão do apontador de pilha (SP) e das instruções de carga e armazenamento do apontador de pilha em memória (LDS e STS).

As instruções lógicas e aritméticas (ADD, ADC, SUB, SBC, NOT, AND, OR, XOR, SHL, SHR, SRA) e as instruções de transferência LDA, LDS e POP afetam apenas os códigos de condição N e Z de acordo com o resultado produzido. As instruções lógicas e aritméticas (ADD, ADC, SUB, SBC, SHL, SHR, SRA) afetam também o código de condição C de acordo com o resultado produzido. As demais instruções (STA, STS, JMP, JN, JP, JZ, JNZ, JC, JNC, JSR, RET, PUSH, IN, OUT, NOP, TRAP e HLT) não alteram os códigos de condição.

A seguir apresentamos uma tabela completa com as instruções do Sapiens e os respectivos códigos de operação.

TABELA I. TABELA DE INSTRUÇÕES

Mnemônico	Código	Descrição
NOP	0000 0000	Não faz nada
STA ender STA @ender	0001 000x	Armazena o acumulador (um byte) na memória
STS ender STS @ender	0001 010x	Armazena o apontador de pilha (dois bytes) na memória
LDA #imed LDA ender LDA @ender	0010 00xx	Carrega o operando (um byte) no acumulador
LDS #imed16 LDS ender LDS @ender	0010 01xx	Carrega o operando (dois bytes) no apontador de pilha (SP)
ADD #imed ADD ender ADD @ender	0011 00xx	Soma o acumulador com o operando (um byte)
ADC #imed ADC ender ADC @ender	0011 01xx	Soma o acumulador com o carry (flag C) e com o operando (um byte)
SUB #imed SUB ender SUB @ender	0011 10xx	Subtrai o acumulador do operando (um byte)
SBC #imed SBC ender SBC @ender	0011 11xx	Subtrai o acumulador do carry (flag C) e do operando (um byte)
OR #imed OR ender OR @ender	0100 00xx	Realiza um "ou" bit a bit entre o acumulador e o operando (um byte)
XOR #imed XOR ender XOR @ender	0100 01xx	Realiza um "ou exclusivo" bit a bit entre o acumulador e o operando (um byte)
AND #imed AND ender AND @ender	0101 00xx	Realiza um "e" bit a bit entre o acumulador e o operando (um byte)
NOT	0110 0000	Complementa ('0' → '1' e '1' → '0') os bits do acumulador.
SHL	0111 0000	Deslocamento do acumulador de um bit para a esquerda, através do *carry*. ´0´s são inseridos á direita.
SHR	0111 0100	Deslocamento do acumulador de um bit para a direita através do *carry*. ´0´s são inseridos á esquerda.
SRA	0111 1000	Deslocamento do acumulador de um bit para a direita através do *carry*, replicando o bit de sinal à esquerda.
JMP ender JMP @ender	1000 000x	Desvia a execução do programa para o endereço
JN ender JN @ender	1001 000x	Desvia a execução do programa para o endereço, apenas se N = 1
JP ender JP @ender	1001 010x	Desvia a execução do programa para o endereço, apenas se N = 0 e Z = 0
JZ ender JZ @ender	1010 000x	Desvia a execução do programa para o endereço, apenas se Z = 1
JNZ ender JNZ @ender	1010 010x	Desvia a execução do programa para o endereço, apenas se Z = 0
JC ender JC @ender	1011 000x	Desvia a execução do programa para o endereço, apenas se C =1
JNC ender JNC @ender	1011 010x	Desvia a execução do programa para o endereço, apenas se C = 0
IN ender8	1100 0000	Carrega no acumulador o valor lido no endereço de E/S
OUT ender8	1100 0100	Descarrega o conteúdo do acumulador no endereço de E/S
JSR ender JSR @ender	1101 000x	Desvia para procedimento
RET	1101 1000	Retorno de procedimento
PUSH	1110 0000	Coloca o conteúdo do acumulador no topo da pilha
POP	1110 0100	Retira o valor que está no topo da pilha e coloca no acumulador
TRAP ender TRAP @ender	1111 000x	Instrução para emulação de rotinas de E/S pelo simulador
HLT	1111 1100	Para a máquina

Foi definida uma linguagem de montagem (assembly language) para este processador obedecendo às regras usualmente encontradas nos sistemas comerciais e compatíveis com a sintaxe previamente utilizada no simulador Neanderwin. A sintaxe completa dos comandos do montador, assim como exemplos mais sofisticados de sua utilização podem ser encontrados na referência [10]. Na maioria são mnemônicos e comandos com sintaxe simplificada e de fácil utilização. A seguir um exemplo de programa em linguagem de montagem para o processador Sapiens:

```
ORG 0
INICIO:
; Faz a leitura do painel de chaves
       IN  0
       STA A
; Se a entrada for 0 termina o programa
       AND 0
       JZ  FIM
; Testa se é par (bit 0=0)
       LDA A
       AND #1
       JNZ SENAO
; Se for, faz pares++
       LDA PARES
       ADD #1
       STA PARES
       JMP INICIO
SENAO:
; Incrementa I
       LDA IMPARES
       ADD #1
       STA IMPARES
       JMP INICIO
FIM:   HLT

ORG 100
A:      DS  1
PARES:  DB  0
IMPARES:DB  0
END 0
```

E. Simulando operações de E/S usando a instrução TRAP

A necessidade de ensinar técnicas de entrada e saída num momento preliminar do ensino, sempre foi uma das mais problemáticas observadas em nossa experiência didática. Mesmo em qualquer problema menos trivial, havia a necessidade de realizar leituras e escritas de dados, porém isso exigia do aluno conhecimentos específicos ainda não aprendidos nas primeiras aulas (laços, testes, *polling*, *flags*, etc.). Desta forma se tornava quase mandatório que os primeiros programas fossem centrados em resolver problemas abstratos usando valores pré-carregados na memória.

Isso poderia ser resolvido, por exemplo, com a inclusão de um pequeno *kernel* pré-carregado em linguagem de máquina na memória do simulador SimuS, que fornecesse diversas funcionalidades (leitura e escrita simplificada, formatação de dados, funções específicas dos dispositivos, etc.). Porém, isso acabaria por tornar ainda mais complexo o processo de ensino, na medida em que professores e alunos ficariam tentados a explorar este código, desfocando o objetivo principal que é o de ensinar arquitetura de computadores. Além disso, a presença deste código pré-definido tornaria ainda mais limitado o uso do recurso já tão escasso como a memória. Optamos então por criar uma abstração, similar àquela apresentada em processadores reais para chamar funções de sistema operacional: a operação TRAP. Em nosso caso, porém, o sistema operacional é algo abstrato, sendo que na verdade essas funcionalidades são implementadas diretamente no simulador.

Neste mecanismo a instrução TRAP é interpretada pelo simulador como um pedido para que ele realize uma determinada operação de E/S. O número do *trap,* ou seja, a função que está sendo solicitada é passada no acumulador. Como o processador Sapiens é carente de registradores, a instrução TRAP tem um operando adicional que é o endereço de memória onde os parâmetros adicionais necessários são passados para o módulo do simulador responsável pela realização da operação. Temos previsão para suportar inicialmente as seguintes operações:

#1 – leitura de um caractere da console. O código ASCII correspondente é colocado no acumulador.

#2 – escrita de um caractere que está no endereço definido pelo operando da instrução TRAP na console.

#3 – leitura de uma linha inteira da console para o endereço definido pelo operando da instrução TRAP.

#4 – escrita de uma linha inteira na console. O operando da instrução TRAP contém o endereço para a cadeia de caracteres que deve ser terminada pelo caractere NULL (0x00). O tamanho máximo da cadeia é de 128 caracteres.

#5 – chama uma rotina de temporização. O operando da instrução TRAP contém o endereço da variável com o tempo a ser esperado em milissegundos.

#6 – chama uma rotina para tocar um tom. A frequência e a duração do tom estão em duas variáveis inteiras no endereço definido pelo operando da instrução TRAP.

#7 – chama uma rotina para retornar um número pseudoaleatório entre 0 e 99 no acumulador. O operando da instrução TRAP tem o endereço com a variável com a semente inicial.

IV. A INTERFACE DE USUÁRIO DO SIMULADOR SIMUS

Mantivemos a interface básica do simulador do Neanderwin, acrescentando mais algumas funcionalidades e agora destacando os módulos de E/S, que estão em janelas que são ativadas apenas quando for conveniente para o usuário. Desde o início do projeto do simulador SimuS nosso objetivo era facilitar ao máximo as atividades didáticas do professor e o apoio mais completo possível para as dificuldades comuns do aluno. Para isso foi criado um ambiente integrado para desenvolvimento, com versões para os sistemas operacionais Windows e Linux, e que inclui os seguintes módulos:

- Editor de textos integrado, que possibilita a abertura, edição e salvamento de arquivo com o código em linguagem de montagem.
- Montador (*assembler*) também integrado, gerando o código objeto final em linguagem de montagem. Possui compatibilidade com programas escritos para o Neander ou Neander-X.
- Simulador da arquitetura, com visualização e modificação dos elementos arquiteturais do processador, tais como registrador de instrução, apontador de instrução, apontador de pilha, acumulador, flags N, Z e C; além da execução passo-a-passo ou direta.
- Módulo de depuração, definindo pontos de parada e variáveis em memória para serem monitoradas, em caso de mudança de valor a execução é interrompida.
- Visualização e modificação da memória simulada, no formato hexadecimal os endereços e também para seu conteúdo, agora expandida para 64 Kbytes.
- Ferramenta de apoio ao aprendizado de instruções, com ajuda integrada ao editor de textos, para a geração de código em linguagem de montagem do processador Sapiens.
- Utilitário para conversão de bases binária, decimal e hexadecimal.
- Simulador de dispositivos de E/S, que inclui o tradicional painel com 8 chaves e visor hexadecimal, acrescidos de um teclado de 12 teclas e um painel de 2 x 16 linhas. Estes dispositivos são acessados no espaço de endereçamento de E/S convencional com instruções de IN e OUT.
- Dispositivos especiais como uma console virtual, acessada pelo mecanismo de TRAP descrito anteriormente. Com esse mecanismo outros módulos mais complexos possam ser facilmente instalados, sem necessidade de expor esta complexidade para os usuários.
- Gerador/carregador de imagem da memória simulada, podendo ser salva em formato hexadecimal. Note que neste caso a compatibilidade

entre o SimuS e o Neanderwin não é mantida, ou seja, a imagem salva em um simulador não pode ser carregada em outro.

A Figura 3 mostra a aparência da tela principal do simulador SimuS. Na parte superior estão o menu geral de operação (Arquivo, Editar, etc.) e diversos botões usados em conjunto com o editor de textos, que seguem o estilo usual de programas com interface gráfica.

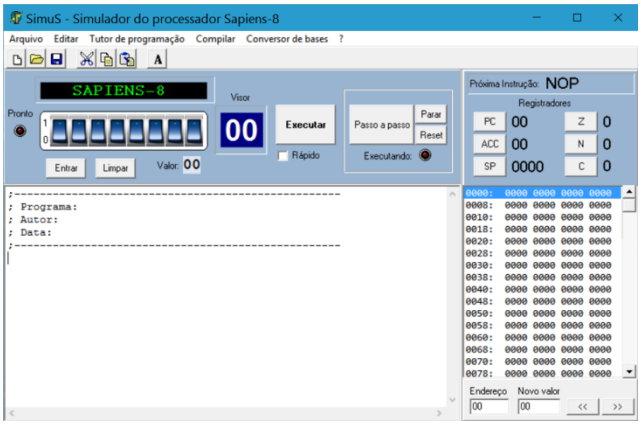

Fig. 3. Tela Principal do Simulador SimuS

Uma vez criado o programa ele será compilado, o que provoca o aparecimento de uma janela pop-up com a listagem, cujo formato de saída é similar à maioria dos montadores profissionais, com indicação dos eventuais erros de compilação. O programa compilado é carregado imediatamente na memória, sendo o conteúdo atualizado e exibido no painel correspondente. Caso se deseje, é possível copiar o conteúdo desta janela para a área de transferência, para colar em algum editor de textos possibilitando eventuais embelezamentos e impressão a posteriori.

O número de instruções do processador Sapiens é relativamente pequeno, mas apesar disso notamos que é importante tornar disponível uma "ajuda online" interativa, que é ativada pelo menu do programa, um sistema de entrada interativa de instruções e pseudo-instruções. A função de criação tutorada de programas faz com que o aluno cometa menos erros e a compreenda melhor o significado das instruções e produza uma sintaxe das instruções mais correta, de forma interativa.

Por último, notamos que para muitos estudantes o domínio das representações hexadecimal e binária, necessário para verificar e alterar a memória, só se dá depois de algum tempo. Mesmo sabendo que as calculadoras do Windows e do Linux exibem resultados em várias bases, incluímos um conversor de bases, que é muito mais simples.

V. CONCLUSÕES

Apresentamos neste artigo o SimuS, simulador para uma arquitetura hipotética do processador Sapiens, uma extensão do processador Neander-X. Entre as diversas novidades que apresentamos podemos destacar uma nova interface de usuário, com novos módulos de E/S e um arquitetura de processador atualizada com novas instruções, como chamada e retorno de procedimento, mantendo ainda compatibilidade com códigos em linguagem de montagem desenvolvidos para o simulador Neanderwin, há mais de 10 anos em uso. Esperamos com essas modificações adicionais estender a possibilidade de uso do simulador SimuS em cursos mais avançados de arquitetura de computadores e programação em linguagem de montagem, permitindo aos alunos e professores explorar o potencial desta nova proposta, de modo a facilitar o ensino e aprendizado de arquitetura de computadores nas universidades do país. O código fonte está disponível em repositório público e os autores se colocam à disposição para apoio em futuras modificações que venham a ser implementadas no simulador SimuS.

Como trabalhos futuros há diversas propostas, que incluem realizar a avaliação do uso da ferramenta em cursos práticos, o que ainda não foi possível, pois o desenvolvimento do SimuS é muito recente. Pretendemos acrescentar melhorias e extensões ao simulador, como uma visão da execução das instruções internamente ao processador; com detalhes da execução em versões com e sem pipeline do processador. Adicionalmente, pretendemos desenvolver um compilador para uma linguagem de alto nível como BASIC, onde a tradução de programas simples em alto nível possa ser visualizada como instruções em linguagem de montagem do processador Sapiens. Pretendemos ainda internacionalizar o simulador, desenvolvendo versões em espanhol e inglês.

REFERÊNCIAS

[1] J. A. S. Borges, G. P. Silva "NeanderWin - um simulador didático para uma arquitetura do tipo acumulador". WEAC, 2006.

[2] R. F. Weber, Fundamentos de Arquitetura de Computadores. 2. Ed. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzatto, 2001.

[3] B. Nikoli, V. Milutinovic, "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization", IEEE Transactions on Education, Vol. 52, No. 4, November 2009

[4] Z. Sridhar, "GNUSim8085, versão 1.3.7", https://launchpad.net/gnusim8085. Acesso em 23 de Junho de 2016

[5] M. Scott, "WinMips64, version 1.57", http://indigo.ie/~mscott/. Acesso em 20 de Julho de 2016.

[6] A. A. Moreira, C. A. P. S. Martins, "R2DSim: simulador didático do RISC reconfigurável". WEAC, pp 9-14, 2009.

[7] A. B. Verona, J. A. Martini, T. L. Gonçalves, "SIMAEAC: Um simulador acadêmico para ensino de arquitetura de computadores". I ENINED - Encontro Nacional de Informática e Educação, pp 424-432, 2009.

[8] E. V. C. L. Borges et al., "SEAC: um simulador online para ensino de arquitetura de computadores" - WEAC, pp. 34-38, 2012

[9] Ziller, R. Microprocessadores: Conceitos Importantes, EEL - UFSC, 2ª edição, Florianópolis, Brasil, 2000

[10] G. P. Silva, J.A.S Borges, O Simulador Neander-X para o Ensino de Arquitetura de Computadores.. Ed. Autor, 75 pp, 2016, disponível em http://www.amazon.com

[11] P. Sanderson, K. Vollmar "MARS Simulator" http://courses.missouristate.edu/KenVollmar/mars/download.htm. Acesso em agosto de 2016.