SASM: uma ferramenta para o ensino do processo de montagem e de conjunto de instruções CISC

Juliano H. Foleiss, Valeria D. Feltrim, Ronaldo A. L. Gonçalves

Departamento de Informática – DIN
Universidade Estadual de Maringá (UEM) – Maringá – PR – Brasil
{juliano.foleiss, valeria.feltrim, ronaldo}@din.uem.br

Abstract. *This paper presents SASM, an assembler developed as part of a simulation environment of the Intel IA-32 architecture, named SOIS. Other than generating machine code for the simulated processor, SASM has educational features that aid the process of learning the language and the assembly process, and facilitate the understanding of the instructions used in CISC architectures.*

Resumo. *Este artigo apresenta o SASM, um montador desenvolvido para integrar um ambiente de simulação da arquitetura IA-32 da Intel, denominado SOIS. Além de gerar código de máquina para o processador simulado, SASM tem características educacionais que auxiliam o processo de aprendizado da linguagem e do processo de montagem, além de facilitar o entendimento das instruções utilizadas em arquiteturas CISC.*

1. Introdução

O projeto SOIS (Sistema Operacional Integrado Simulado), iniciado em 2004 [Gonçalves, 2004], visa o desenvolvimento de um ambiente de simulação que integra Compilador, Montador, Sistema Operacional [Assunção, 2008], Processador/Memória [Cruz, 2008] e Sistema de E/S [Cruz, 2007], possibilitando criar, executar e depurar programas sob o controle de um sistema operacional multiprogramado. SOIS pode ser configurado para operar de diferentes formas e mostrar passo a passo suas operações internas e estatísticas de execução. Desta forma, SOIS pode ser usado tanto para o ensino de graduação em computação quanto para avaliação de desempenho de sistemas computacionais em proposição. O foco deste trabalho está em evidenciar as facilidades disponibilizadas pelo montador.

O montador do ambiente SOIS, denominado SASM (SOIS *Assembler*), toma como entrada um programa escrito em um subconjunto da linguagem de montagem padrão da Intel (aceita pelo processador SOIS) e gera um arquivo com código-objeto da arquitetura IA-32 [Intel, 2004] em formato hexadecimal. Mediante opções simples na linha de comando, o montador pode ser configurado para guiar o usuário por todo o processo de montagem e a interação entre seus componentes, evidenciando seu trabalho em cada linha do código fonte: detalhando o processo de reconhecimento dos componentes da instrução, seu formato e o processo de geração do código-final. Tudo isso é feito com a exibição de mensagens objetivas e claras, que permitem que o foco esteja no entendimento do conjunto de instruções e sua estrutura.

Na saída, ao lado de cada instrução hexadecimal, pode ser colocado o código de montagem responsável por aquela instrução. Isso permite ao usuário verificar exatamente como cada instrução de montagem é traduzida em instruções de máquina, facilitando a visualização do formato de instrução de uma arquitetura alvo. Além disso, SASM reconhece constantes, saltos simbólicos, cálculo de deslocamento misto (envolvendo símbolos e valores imediatos) e inicialização de regiões de memória com valores pré-determinados. Para facilitar o processo de depuração, o montador permite efetuar o *dump* da tabela de símbolos, mostrando todos os símbolos definidos e seus respectivos endereços. Este processo conta com mensagens de erros de fácil compreensão pelo programador, agilizando a aprendizagem da linguagem de montagem de forma objetiva, diminuindo o tempo gasto com depuração.

Este trabalho está organizado da seguinte forma. Na Seção 2 são apresentados os objetivos principais do SASM. Trabalhos relacionados e as relações com o SASM são descritos na Seção 3. As funcionalidades do SASM são apresentadas na Seção 4. A Seção 5 apresenta a sintaxe do SASM. Na Seção 6 é apresentado como o SASM pode ser utilizado como ferramenta didática por meio de um estudo de caso. Finalmente, na Seção 7, são apresentadas as conclusões e previsões de trabalhos futuros.

2. Objetivos do SASM

O SASM tem dois objetivos principais. O primeiro é o de permitir a montagem de programas em linguagem de montagem, gerando código-objeto compatível IA-32. Esta facilidade possibilita escrever programas que explorem os recursos do sistema simulado da forma desejada. O segundo objetivo é o de auxiliar no ensino de aspectos relacionados ao processo de montagem, envolvendo o ensino de linguagem de máquina, da linguagem de montagem, do processo de montagem e do conjunto de instruções. O fato da arquitetura IA-32 ser predominantemente CISC é útil para mostrar a complexidade da linguagem de máquina e do conjunto de instruções [Stallings, 2002].

3. Trabalhos relacionados

Diversos trabalhos têm sido desenvolvidos para auxiliar no ensino da linguagem de montagem, principalmente para arquiteturas RISC. O MARS [Vollmar, 2006] é um simulador da arquitetura MIPS com ênfase na fase de depuração. Ele permite que variáveis e registradores sejam alterados a qualquer momento, além de permitir a execução passo-a-passo das instruções. Outro ponto forte do MARS é que ele permite que sejam ligados *breakpoints* em qualquer linha de código, o que beneficia o processo de depuração. O ambiente disponibiliza um editor básico de código e permite a criação de novas instruções para o simulador. O WebMips [Giorgi, 2004] também é um simulador da arquitetura MIPS, que possui uma interface web detalhada, com um depurador mostrando todo o processo de execução do código em termos de pipeline. O formato das instruções MIPS também é tratado com bastante cuidado, sendo explicitados os diferentes formatos e seus componentes para cada instrução traduzida.

Outro simulador MIPS é o MipSim [Koka, 2009], que possui um depurador passo-a-passo e foca seus esforços principalmente em seu editor, que possui características avançadas, como completador de código e marcador de sintaxe. Um editor poderoso também é de grande ajuda no aprendizado da linguagem de montagem,

já que em tempo de edição o código pode ser verificado pelo autor. O P88 *Assembler and Simulator* [Ogihara, 2007] compõe-se de montador e simulador para a arquitetura hipotética P88, a qual foi projetada originalmente para servir como uma arquitetura de fácil assimilação para o ensino de linguagens de montagem, arquitetura de computadores e compiladores. Ogihara projetou um simulador robusto para uma linguagem de montagem simplificada que ajuda no entendimento do desenvolvimento de programas em baixo-nível.

Muitos trabalhos foram desenvolvidos focando fortemente no aspecto de depuração de programas escritos em linguagem de montagem, entretanto, todos voltados para arquiteturas com conjunto reduzido de instruções (RISC), assim como o Mipsasm [Asanović, 2003]. Trabalhar com instruções CISC é muito mais difícil e as máquinas existentes são na maioria desta classe. Nenhum dos trabalhos aqui citados faz o detalhamento do processo de montagem com a riqueza de informações que o SASM faz, muito embora o Mipsasm também permita um bom nível de detalhamento. SASM é o único dos trabalhos citados que possui um modo de montagem passo-a-passo, permitindo que o aluno veja a tradução do programa linha-a-linha.

4. O montador SASM

SASM foi escrito em C++, sob a plataforma GNU/Linux, utilizando o paradigma de orientação a objetos. O SASM implementa um subconjunto da linguagem de montagem da Intel, aceito pelo processador SOIS. A funcionalidade principal do SASM é a geração de código-objeto, que pode ser representado em dois formatos. O primeiro é o formato hexadecimal, que pode ser acompanhado ou não de suas respectivas instruções em linguagem de montagem. O segundo é o formato binário, pronto para ser executado.

A interface do montador é em linha de comando, simples e fácil de usar, acompanhada de um sistema intrínseco de ajuda, que pode ser ativado por meio da opção -?. O montador também conta com mensagens de erros e avisos, ajudando no processo de depuração e facilitando a aprendizagem da linguagem de montagem. SASM permite gerar, a partir de uma opção na linha de comando, um arquivo contendo o *dump* da tabela de símbolos, que auxilia a localização das variáveis no programa binário. Esta simplicidade delega ao SASM fortes características educacionais.

O cerne educacional do SASM é o modo de depuração (também conhecido como *VERBOSE* ou *DEBUG*), que detalha o processo de montagem passo-a-passo, permitindo ao usuário ver todas as interações entre os componentes do montador, além de evidenciar todas as etapas do processo de montagem. O nível de verbose pode ser escolhido por meio da opção –v. Existem quatro níveis de verbose (0, 1, 2 e 3), sendo 0 o nível default. Quanto maior o nível, maior a quantidade de detalhes a ser mostrada. Cada nível de verbose herda as mensagens dos níveis anteriores, o que significa que se o nível 3 for selecionado, também serão mostradas as mensagens dos níveis 1 e 2.

No nível 0, o montador trabalha silenciosamente, emitindo apenas mensagens de erro e avisos. No nível 1, são mostradas as etapas que o montador está executando em um dado momento. No nível 2, são mostrados detalhes sobre a análise léxica e semântica do programa sendo montado. Esse nível também mostra os cálculos dos endereços de todos os símbolos (a partir do cálculo do tamanho do código-objeto gerado para cada instrução), bem como a inserção de cada um na tabela de símbolos. Ainda

neste nível, ao término da fase de reconhecimento de símbolos, é exibida a tabela de símbolos, informando seus respectivos endereços, valores e tipos.

O nível 3 apresenta mais detalhes sobre a análise léxica e semântica, mostrando também os modificadores e formatos (deslocamentos, valores imediatos e combinação de operandos) que determinam o tamanho do código-objeto gerado a partir de cada instrução. Neste nível, a fase de codificação é detalhada ao máximo, mostrando o reconhecimento do formato da instrução, a codificação do byte de endereçamento (BYTE MOD/RM) e o código-objeto gerado a partir de cada instrução. Para permitir avaliar o perfil do código de montagem e do código de máquina gerado, algumas informações, tais como a quantidade de cada tipo de instrução traduzida e o tamanho em bytes do código e dos dados, são mensurados em tempo de montagem para a geração de estatísticas que podem ser visualizadas *post mortem* com a opção -e.

5. Sintaxe do SASM

A Intel definiu um padrão de linguagem de montagem consistente que é utilizado em todos os documentos relacionados à arquitetura IA-32 [Intel, 2004]. O montador SASM implementa um reconhecedor para um subconjunto dessa linguagem. O formato padrão da instrução de montagem reconhecida pelo SASM é o seguinte:

RÓTULO: MODIFICADOR MNEMÔNICO OP1, OP2 ;comentário

onde RÓTULO é o nome simbólico da posição de memória, MODIFICADOR são os símbolos que representam uma mudança no prefixo da instrução, MNEMÔNICO é o nome que representa códigos de operação que tem uma mesma função e OP1 e OP2 são os operandos que podem ou não ser usados, dependendo do código de operação. Por convenção, quando usados, OP1 é o operando destino e OP2 é o operando fonte. Os operandos de uma instrução SASM podem ser:

- Valores imediatos: expressos nas bases octal, decimal e hexadecimal, 20o, 16 e 0Ah respectivamente, além de uma constante simbólica, tal como OFFSET;
- Registradores de propósito-geral: de 8-bits (AL, AH, BL, BH, CL, CH, DL, DH), 16-bits (AX, BX, CX, DX, SI, DI, BP, SP) ou 32-bits: (EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP)
- Expressões de memória (envolvidas por colchetes []), podendo ser a) com deslocamento imediato, que seguem o mesmo formato para valores imediatos, por exemplo, [15h], [MATRIZ_A] e [4], ou b) com registrador-base mais deslocamento imediato, por exemplo, [EAX], [EBP+15h], [EBP+MATRIZ_A], [MATRIZ_A +10] e [ESP+MATRIZ_A+15h].

O MODIFICADOR está relacionado ao tamanho do operando. Seu uso só é necessário quando há um acesso à memória e o montador não consegue determinar o tamanho da instrução apenas pela análise dos operandos. Tal situação normalmente ocorre quando há uma instrução na qual um dos operandos é uma expressão de memória e o outro operando é um valor imediato. Vale lembrar que não é possível que ambos os operandos sejam expressões de memória em uma mesma instrução. O modificador pode ser um dos seguintes valores: BYTE, WORD e DWORD, que forçam certo operando a ser tratado como sendo de 8, 16 ou 32-bits, respectivamente.

O subconjunto da linguagem de montagem da Intel aceita pelo SASM implementa os modos de endereçamento imediato, operando-registrador e base com deslocamento. Foram implementadas 33 classes de mnemônicos no SASM. Devido à complexidade das instruções, existem diversos formatos de instrução, no que se refere aos tipos dos operandos e suas combinações.

A classe de operações básicas da ULA (ADD, OR, XOR, AND, SUB, CMP) segue o formato MNEMONICO (r/m32, imd32 | r/m32, r32 | r32, r/m32). As operações de pilha (PUSH, POP) e de incremento e decremento (INC, DEC) têm o formato MNEMÔNICO (r32). Operações complexas da ULA (DIV, IMUL, MUL) têm o formato MNEMÔNICO (r/m32). Instruções de controle de fluxo estático, como HLT e RET, não possuem operando. A instrução CALL e a de desvio incondicional tem o formato MNEMÔNICO (r/m32 | rel32). Os desvios condicionais (JE/JZ, JG, JL, JNE/JNZ) seguem o formato MNEMÔNICO (rel32). As instruções de deslocamento (SAR, SAL/SHL, SHR) e de rotação (ROL, ROR, RCL, RCR) têm o formato MNEMÔNICO (r/m32, CL | r/m32, imd8). As instruções de entrada e saída (IN, OUT) seguem dois formatos diferentes: IN AL, DX e OUT DX, AL. A instrução mais complexa em termos de formato é a instrução de transferência de dados (MOV). Dentre os inúmeros formatos para essa instrução, os seguintes foram implementados: MOV (r32, imd32 | r/m32, r32 | r32, r/m32 | r/m8, r8, | r8, r/m8).

Além dos mnemônicos, foram definidas quatro diretivas de montagem que permitem a criação de constantes simbólicas ou a reserva de posições de memória, sendo possível a inicialização dessas diretamente na linha de sua definição. O formato para as instruções de definições de reserva de posições de memória (DB, DW, DD) é o seguinte: DIRETIVA (imd, [lista_de_inicialização]), onde lista_de_inicialização é opcional e segue a forma $\{a_1, a_2, ..., a_n\}$, onde a_k é uma constante (não simbólica) automaticamente convertida para o tamanho sendo reservado. Caso a lista de inicialização seja omitida, todos os valores são inicializados com 0. Constantes simbólicas são definidas com a diretiva EQU, que segue o formato EQU (imd), onde o nome da constante é o rótulo da linha da instrução e o valor é imd. Em ambos os casos, imd é uma constante qualquer (simbólica ou não). Alguns exemplos de utilização das instruções e das diretivas são mostrados na Figura 1.

```
;exemplos de diretivas

tam_v_b: equ 0Fh   ;constante simbólica
i: DB 1
j: DD 1, {5h}
vetor_a: DD 10, {1, 2, 3, 4, 5, 6, 7, 8, 9, 0Ah]
vetor b: dd tam_v_b

;exemplos de instruções
;rótulo   instrução              ;formato da instrução

inicio: MOV EAX, ECX            ;MOV r32, r/m32
        MOV [BH], CL            ;MOV r/m8, r8
        ADD [ecx+10h], edx      ;ADD r/m32, r32
        INC ECX                 ;INC r32
        OUT DX, AL              ;OUT DX, AL
        CMP ECX, [EAX]          ;CMP r32, r/m32
        JNE inicio              ;JNE rel32
        RET                     ;RET
```

Figura 1. Exemplos de diretivas e instruções aceitas pelo SASM

6. Utilizando o SASM como ferramenta de ensino

Para mostrar como o SASM pode ser utilizado como ferramenta de ensino, um estudo de caso foi utilizado na montagem e depuração do programa exemplo mostrado na Figura 2. Embora programas maiores e mais complexos possam ser montados pelo SASM, esse programa foi escolhido por conter diversos elementos que exemplificam o tipo de ajuda fornecida pela ferramenta.

```
;Este programa acumula os valores do vetor em eax
num_valores:    equ 4
vetor:          dd num_valores, {10h, 21, FFh, 0Ah}
mov edx, num_valores
shl edx, 2
mov ecx, [edx+vetor]
mov eax, 0h
inicio:
   sub ecx, 4
   add eax, [ecx]
   jnz inicio
hlt
```

Figura 2. Programa-exemplo

Os exemplos apresentados a seguir foram gerados utilizando-se o modo de verbose nível 3. A Figura 3 mostra a inicialização do montador, bem como a transição para a primeira fase do processo de montagem: a fase de reconhecimento de símbolos. As linhas iniciadas por V-NÍVELX indicam as informações de verbose que são mostradas no nível X.

```
juliano@ubuntu-fx:~/SOIS/sasm/codeblocks/sasm$ ./sasm exemplo.asm -bce
Executando o SASM com nível 3 de verbose

V-NÍVEL1: Incluindo instruções e diretivas em seus vetores correspondentes
V-NÍVEL1: Inicializando a tabela de símbolos...

V-NÍVEL1: Iniciando a fase 1/2 (Reconhecimento dos Símbolos)...

------------------------------------------------------------
V-NÍVEL2: Linha 1: num_valores:    equ 4
V-NÍVEL2: Linha normalizada: NUM_VALORES: EQU 4

V-NÍVEL2: COMPONENTES DA INSTRUÇÃO:
Rótulo: >NUM_VALORES<
Instrução: >EQU<
Op1: >4<
Op2: >4<
Formato:
   OP1: 38: | IMEDIATO |
   OP2: 38: | IMEDIATO |
V-NÍVEL2: Inserindo constante na tabela de símbolos: NUM_VALORES = 4 (4h)
V-NÍVEL2: Diretiva EQU tokenizada!
V-NÍVEL2: Endereço atual: 0 (0h)
```

Figura 3. Início da montagem

Ainda na Figura 3, é possível ver o reconhecimento do símbolo NUM_VALORES, utilizando-se a diretiva EQU. As mensagens de verbose mostram detalhadamente o formato da instrução reconhecida (nesse caso uma diretiva de montagem), além de informações específicas a essa instrução. Nesse caso, uma constante é inserida na tabela de símbolos e o endereço do código sendo gerado não é modificado, já que constantes não geram efetivamente nenhum código-objeto.

A Figura 4 apresenta a montagem de duas instruções MOV. Sabe-se [Stallings, 2002] que esse tipo de instrução é o mais utilizado em programas, por isso, a Intel criou diversas instruções específicas para transferência de dados com o intuito de especializá-las a fim de obter melhor desempenho para cada situação. O SASM mostra como essas instruções, mesmo sendo de transferência de dados, possuem códigos-objeto distintos.

```
V-NÍVEL2: Linha 5:     mov ecx, [edx+vetor]
V-NÍVEL2: Linha normalizada: MOV ECX, [EDX+VETOR]
V-NIVEL2: COMPONENTES DA INSTRUÇAO:
Rótulo: ><
Instrução: >MOV<
Op1: >ECX<
Op2: >[EDX+VETOR]<
Formato:
   OP1: 4: | REGISTRADOR 32-BITS |
   OP2: 801C0: | OP. MEMÓRIA | DESLOCAMENTO C/ SÍMBOLO |
V-NíVEL3: Deslocamento de 32-bits
V-NÍVEL2: Tamanho do código-objeto: 6 bytes
V-NÍVEL3: MOV REG, MEM
V-NÍVEL3: Byte MOD/RM: 8A = (MOD: 2 << 6 | OP: 1 << 3 | RM: 2)
V-NIVEL3: > MOV ECX, [EDX+VETOR]<: Código-objeto: 8B 8A 00000000(d) (i)
V-NÍVEL2: Endereço atual: 30 (1Eh)
-----------------------------------------------------
V-NÍVEL2: Linha 6:     mov eax, 0h
V-NÍVEL2: Linha normalizada: MOV EAX, 0H
V-NIVEL2: COMPONENTES DA INSTRUÇAO:
Rótulo: ><
Instrução: >MOV<
Op1: >EAX<
Op2: >0H<
Formato:
   OP1: 4: | REGISTRADOR 32-BITS |
   OP2: 38: | IMEDIATO |
V-NíVEL3: Imediato de 32-bits
V-NÍVEL2: Tamanho do código-objeto: 6 bytes
V-NíVEL3: Operação MOV REG, IMD não possui o byte MOD/RM
V-NÍVEL2: Tamanho do código-objeto: 5 bytes
V-NÍVEL3: MOV REG, IMD
V-NÍVEL3: > MOV EAX, 0H<: Código-objeto: B8 (d) 00000000(i)
V-NÍVEL2: Endereço atual: 35 (23h)
```

Figura 4. Comparação de instruções MOV

Na Figura 4, a linha 5 tem uma instrução no formato de instrução “MOV REG, MEM”, que necessita de um byte de endereçamento (o byte MOD/RM) e 4 bytes para o deslocamento, ocupando assim 6 bytes. Também é interessante notar que seu código de operação é 8B. Já na linha 6, que também tem uma instrução de transferência de dados, o byte MOD/RM não é necessário, já que existe um código de operação específico para o formato MOV EAX, IMD (nesse caso B8), levando a um tamanho de instrução de

apenas 5 bytes. Neste exemplo fica explícita a complexidade do conjunto de instruções da arquitetura IA-32, predominantemente CISC, e como o detalhamento do processo de montagem pode auxiliar no entendimento das instruções.

Para finalizar, a Figura 5 mostra como o SASM detalha o cálculo dos endereços das instruções de saltos. É importante lembrar que os saltos implementados no SOIS são todos relativos, contrastando com saltos absolutos. Além disso, a Figura 5 também mostra a característica da disposição dos valores de múltiplos bytes na memória, neste caso, *little-endian*. Neste exemplo, o salto calculado é -14 (FFFFFFF2h), sendo que o montador converteu para *little-endian* e gerou o código com o operando F2FFFFFF.

```
V-NÍVEL2: Linha 10:     jnz inicio
V-NÍVEL2: Linha normalizada: JNZ INICIO

V-NÍVEL2: COMPONENTES DA INSTRUÇÃO:
Rótulo: ><
Instrução: >JNZ<
Op1: >INICIO<
Op2: ><
Formato:
   OP1: 10038: | SÍMBOLO | IMEDIATO |
   OP2: 1000: | INEXISTENTE |
V-NÍVEL2: Tamanho do código-objeto: 6 bytes
V-NÍVEL3: JNZ IMD
V-NÍVEL3: Salto condicional JNZ
          Posição atual: 49 (31h)
          Destino (INICIO): 35 (23h)
          Salto: -14 (FFFFFFF2h)
V-NÍVEL3: > JNZ INICIO<: Código-objeto: 0F 85 F2FFFFFF(i)
V-NÍVEL2: Endereço atual: 49 (31h)
```

Figura 5. Cálculo de salto

7. Conclusões e Trabalhos Futuros

Além de facilitar a escrita de programas mais complexos e longos sejam escritos para serem executados pelo SOIS, quando comparado com a versão anterior do SOIS em que os executáveis eram gerados manualmente, o montador SASM também pode ser utilizado como uma ferramenta educacional no ensino de conceitos relacionados à linguagens de montagem e arquitetura de computadores. O modo de depuração do SASM permite que o aluno visualize, passo-a-passo, todo o processo de montagem, desde a análise léxica até a fase de codificação.

O detalhamento de todos os passos, com precisão e de forma completa e objetiva, permite ao aluno focar no processo de montagem, por meio de uma interface simples e objetiva. A exibição do código de montagem ao lado do código de máquina hexadecimal no arquivo de saída também ajuda na compreensão da linguagem de máquina utilizada pelo IA-32, evidenciando as características e peculiaridades do código em arquiteturas CISC. Facilidades como o *dump* da tabela de símbolos e mensagens de erros simplificam a depuração de programas e ajudam no processo de aprendizagem.

O estudo de caso comprova a relevância do montador. Entretanto, a ferramenta será utilizada sistematicamente junto aos alunos de graduação para que uma avaliação

mais rigorosa seja feita. Atualmente, o compilador SOIS está em desenvolvimento. Denominado SCC (SOIS C Compiler), o compilador irá gerar código *assembly* compatível com o SASM, permitindo assim a escrita de programas em linguagem de alto-nível para serem executadas no ambiente simulado, devendo também ser direcionado ao ensino de graduação. Modos de depuração e outras facilidades para observação do processo de compilação também estão previstas como trabalhos futuros.

8. Referências Bibliográficas

Assunção, G. P., Gonçalves, R. A. L. Simulador SOIS - Módulo Sistema Operacional: Proposta de uma ferramenta de auxílio ao ensino da disciplina de Sistemas Operacionais In: VIII FITEM/ XI Mostra de Trabalhos de Informática, p.70-81, Maringá, 2008.

Asanović, K. Mipsasm, 2003. http://inst.eecs.berkeley.edu/~cs152/mipsasm/; acessado em 15 de maio de 2009.

Cruz, E. H. M.; Silva, Valdemir P.; Gonçalves, R. A. L. Sistema Operacional Integrado Simulado: Módulo de Entrada e Saída. In: ERI - Escola Regional de Informática, Guarapuava: Unicentro / SBC, p.89 – 98. 2007.

Cruz, E. H. M., Foleiss, J. H., Assunção, G. P., Gonçalves, R. A. L. SSSim - Simulador Funcional Detalhado de Processador Superescalar Baseado em Ciclos de Execução Real: Uso na Avaliação de Desempenho In: WSCAD-Concurso de Trabalhos de Iniciação Científica, Campo Grande. p.1 – 4, 2008.

Giorgi, R., Branovic, I., Martinelli, E. "WebMIPS: a new web-based MIPS simulation environment for computer architecture education". In: *Proceedings of the 2004 workshop on Computer Architecture education*, Munique, Alemanha, 2004.

Gonçalves, R. A.; Mulati, M. H.; Silva, V. P. da; Gonçalves, Ronaldo A. L. Sistema Operacional Simulado: Ferramenta para o Ensino de Graduação. In: XXIV CSBC/XII WEI - Workshop de Educação em Informática, Salvador, 2004.

Intel Pentium® Processor Architecture Software Developer's Manual, V1, 2 e 3: Basic Architecture, Instruction Set Reference e System Programming Guide. Intel Corporation. 2004.

Koca, Ç. Mipsim – MIPS Assembly Language Simulator. http://www.mipsim.com/ mipsim/; acessado em 15 de maio de 2009.

Ogihara, T. "*P88 Assembler and Simulator*". http://homepage.mac.com/t_ogihara/ software/CLI/p88/index.html; acessado em 15 de maio de 2009.

Stallings W. Arquitetura e Organização de Computadores: Projeto para o Desempenho, 5ed. São Paulo: Editora Prentice Hall, 2002.

Vollmar, K., Sanderson, P. *"MARS: An Education-Oriented MIPS Assembly Language Simulator"*. In: *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pp. 239-243, Houston, EUA, 2006.