RBECT
Revista Brasileira de Ensino de Ciência e Tecnologia

Revista Brasileira de Ensino de Ciência e Tecnologia
ISSN: 1982-873X

https://periodicos.utfpr.edu.br/rbect

Um simulador de apoio ao ensino de linguagem de montagem

Rene Pegoraro
rene.pegoraro@unesp.br
0000-0003-0314-8660
Universidade Estadual Paulista, Bauru, São Paulo, Brasil.

Marcelo Nicoletti Franchin
marcelo.franchin@unesp.br
0000-0003-3021-9874
Universidade Estadual Paulista, Bauru, São Paulo, Brasil.

RESUMO

Este artigo descreve um simulador de um processador hipotético simples para a introdução de conceitos de linguagem de montagem em alunos do ensino médio e nível superior. O simulador, desenvolvido para ser utilizado como uma ferramenta didática, oferece ao aluno, em uma interface gráfica, um modelo do funcionamento de um computador do ponto de vista da programação de baixo nível. Nesta ferramenta, o usuário carrega o seu programa em linguagem de máquina e visualiza as mudanças decorrentes da sua execução na memória e nos registradores. Considerando a importância do entendimento da diferença entre linguagem de montagem e linguagem de máquina, o aluno é orientado a escrever seu código em linguagem de montagem e em seguida, fazer a montagem para obter a linguagem de máquina. No início, este procedimento é de forma manual e posteriormente através de um programa montador. A montagem manual esclarece alguns conceitos, relacionados à geração de código executável, escondidos nos ambientes de desenvolvimento integrados. Apesar da ferramenta simular um processador hipotético simplificado, ela foi construída seguindo a sintaxe de instruções usada na arquitetura Intel de 32 *bits* (IA-32), permitindo que o aluno utilize os conceitos absorvidos quase diretamente no entendimento de outras linguagens de montagem em computadores reais. Esta ferramenta é utilizada introdutoriamente na disciplina Linguagem de Montagem do Curso de Ciência da Computação na Universidade Estadual Paulista em Bauru com significativo aumento da taxa de aprovação dos alunos, a qual pode ser confirmada a partir dos dados apresentados sobre oito anos lecionados, sendo quatro deles com o uso do simulador.

PALAVRAS-CHAVE: Linguagem de Montagem. Simulador. Ensino de Computação.

INTRODUÇÃO

Grande parte dos estudantes e profissionais tem preferência por ferramentas tecnológicas que facilitam e agilizam os trabalhos de desenvolvimento na área de computação, porém, muitas vezes, ignoram que estas tecnologias ocultam elementos básicos que precisaram ser desenvolvidos para suportá-las e que a compreensão destes elementos pode melhorar o uso de tais ferramentas. Assim, apesar desta preferência à programação de alto nível, bibliotecas especializadas, interfaces homem-máquina, mecanismos de abstração de dados e da ênfase ao entendimento dos computadores como máquinas representadas por modelos abstratos, a linguagem de montagem mantém a sua importância, pois é onde são construídos muitos dos mecanismos dentro destas ferramentas. Ela aparece em disciplinas como: i) Introdução à Computação, relacionando linguagem de alto nível com linguagem de montagem e linguagem de máquina, codificação de dados e mecanismos de chamada e retorno de sub-rotinas; ii) Compiladores, transformando linguagens de programação de alto nível em linguagem de montagem ou de máquina; iii) Arquitetura e Organização de Computadores, partindo da organização até a execução da linguagem de máquina na arquitetura; e iv) Sistemas Operacionais, em situações em que são necessárias instruções de baixo nível, programadas com linguagem de montagem, onde não é possível obter alto desempenho de execução na codificação em linguagens de alto nível, como a troca de contexto de processos ou chamadas a serviços do sistema. Esta importância é reforçada, pois a linguagem de montagem figura nos guias de currículos da *Association for Computing Machinery* (ACM) para cursos de graduação em Ciências da Computação (ACM/IEEE, 2013) e Engenharia da Computação (ACM/IEEE, 2016).

Normalmente, a disciplina de Linguagem de Montagem é ministrada no segundo ou terceiro semestre dos cursos de computação, após um semestre de aprendizado de programação em alguma linguagem de alto nível. Apesar da complexidade desta disciplina, o seu ensino não deve desencorajar ou desmotivar o aluno iniciante na área da computação.

Neste sentido, ferramentas de programação podem ser utilizadas para simplificar o aprendizado, entusiasmar ou motivar os estudantes. Ferramentas como ambientes integrados de programação com a visualização de registradores e seções da memória do processador, programação de microcontroladores, simuladores e emuladores de dispositivos reais complexos ou hipotéticos, são usadas no ensino, mas, frequentemente, não tem o foco na simplicidade do entendimento do funcionamento de conceitos da programação de processadores em baixo nível desejáveis numa ferramenta de ensino.

Este artigo apresenta o AS.SIM (*ASsembly SIMulator*), usado para introduzir aos alunos os fundamentos de programação em linguagem de montagem e linguagem de máquina. Ele é composto por quatro registradores, dois indicadores (*flags*), três botões de controle e 256 posições de memória; apresentados concomitantemente em uma única janela fixa, configurando um modelo simplificado de um processador.

O restante deste trabalho está organizado como segue: na seção 2, uma revisão de alguns simuladores usados para o ensino de linguagem de montagem; na seção 3, as características do simulador AS.SIM e sua utilização; na seção 4, a abordagem empregada no curso de Ciência da Computação da Unesp de Bauru; na

seção 5, os resultados obtidos com a utilização desta ferramenta e finalmente são apresentadas as conclusões.

FERRAMENTAS PARA O ENSINO DE LINGUAGEM DE MONTAGEM

Existe uma ampla gama de ferramentas que podem ser utilizadas para o ensino de linguagem de montagem, indo desde ambientes voltados ao desenvolvimento e teste profissionais de programas em processadores reais, até simuladores de processadores hipotéticos destinados ao ensino de arquitetura de computadores e linguagem de montagem.

Com o foco no ensino de linguagem de montagem, é desejável que a ferramenta possibilite a um aluno com pouca experiência o aprendizado de uma arquitetura simples que seja base para a programação de um processador real, assim como a interpretação simples dos dados que estão sendo apresentados, a facilidade de uso, a disponibilidade nos sistemas operacionais comuns aos alunos dos cursos de computação e a visualização da execução passo a passo das instruções em um programa. Ambientes complexos com várias janelas e informações desnecessárias a um aluno iniciante podem levá-lo a tentar entender informações pouco relevantes para o estágio inicial, desviando a sua atenção e dificultando o aprendizado no ponto de interesse. Neste sentido, uma interface simples que apresente apenas a memória, os registradores e *flags* primordiais faz com que o aluno se foque na compreensão da programação em baixo nível. Assim, para o aluno iniciante, descartam-se os ambientes de desenvolvimento profissionais, restando os simuladores de arquiteturas mais simples.

É desejável que o conhecimento adquirido inicialmente em uma ferramenta não se perca quando o aluno avança no seu aprendizado, que a arquitetura utilizada seja introdutória para uma das arquiteturas mais comuns ao curso onde este aluno está inserido e que a ferramenta acompanhe parte de uma arquitetura real existente. Assim, para o AS.SIM, optou-se pela Arquitetura Intel de 32 *bits* (IA-32) (INTEL, 2016).

Nikolic *et al*. (2009) comparam 28 simuladores, e, apesar de praticamente todos possibilitarem a programação em linguagem de máquina, não se aborda a didática destes simuladores na programação em linguagem de montagem. Esmeraldo *et al*. (2019) comparam 16 simuladores utilizando 7 métricas, incluindo uma específica para linguagem de montagem. Destes, foram escolhidos pela simplicidade ao aluno iniciante o CompSim, MarieSim, SIMAEAC, SimuS.

O CompSim (ESMERALDO; LISBOA, 2017) inclui uma interface gráfica integrada a uma plataforma de *hardware* simulável, que permite simplificar a configuração dos componentes de *hardware*, programação do processador, execução e avaliação de desempenho de novos sistemas computacionais. Na janela de programação de baixo nível, pode-se ver os registradores internos da CPU, a memória cache, parte da memória principal, e o código da linguagem de montagem. Alguns registradores podem receber acesso através de posições de memória. O endereçamento indireto à memória acontece apenas pelas instruções STI e LDI, onde uma posição de memória indica a posição a ser manipulada. E apenas dois tipos de saltos, JN (salta se negativo) e JZ (salta se zero).

A MARIE, do acrônimo *Machine Architecture that is Really Intuitive and Easy*, foi desenvolvida para o entendimento de conceitos básicos de uma arquitetura

von Neumann totalmente funcional com um conjunto descomplicado de instruções. A MARIE é base para o simulador MarieSim. Todos os componentes do sistema, incluindo registros, instruções e memória, são visíveis na tela simultaneamente. As 13 instruções têm tamanho fixo em 2 *bytes* e são específicas à MARIE. Na janela principal, pode-se ver os registradores internos da CPU, parte dos 4k da memória principal, e o código da linguagem de montagem. Diferente de muitas arquiteturas convencionais, não utiliza *flags*; e os desvios consideram o valor do acumulador sendo menor, igual ou maior que zero (NULL; LOBUR, 2003).

O SIMAEAC, Simulador Acadêmico para Ensino de Arquitetura de Computadores, implementa a arquitetura do processador 8085, por meio de instruções lógicas, aritméticas, de manipulação de pilha, de chamada e retorno de sub-rotinas, de desvios condicionais e incondicionais, e de transferência de dados entre registradores e entre registradores e memória. Exibe os registradores A (Acumulador), B, C, D, E, H, L, PC e SP, todos de 8 *bits*, e *flags*, de 5 *bits*. Área de memória reduzida, de 00 até 8F de endereçamento, é inteiramente apresentada na interface sem a necessidade de rolagem da barra lateral. Dessa forma, a visualização da célula em que se encontra a instrução sendo executada está sempre disponível. Com essa alteração, instruções que manipulam memória, além dos registradores SP e PC, devem manipular 8 *bits* (VERONA; MARTINI; GONÇALVES, 2009).

Silva e Borges (2016) apresentam o simulador SimuS e o processador hipotético Sapiens que é a evolução do Neander-X que apresenta 31 instruções com até 3 *bytes* cada. Dentre muitas, as principais evoluções são os 64K de memória, o endereçamento indireto, a pilha, e instruções específicas para chamada e retorno a sub-rotinas. A interface é simples: a tela principal apresenta uma área para a digitação da linguagem de montagem; botões para executar, passo a passo e parar; os registradores; os *flags*; parte dos 64k da memória; e a entrada e saída. A carga do programa na memória acontece diretamente pela montagem do código digitado em linguagem de montagem. Weber (2001 apud SILVA; BORGES, 2016) define a máquina Neander sendo uma arquitetura rudimentar baseada em acumulador com 256 *bytes* de RAM, instruções com até 2 *bytes*, modos direto, 16 instruções.

Muitos outros trabalhos voltados a simuladores específicos para ensino-aprendizagem podem ser encontrados na literatura. Alguns estão citados nesta seção.

Silva e Borges (2018) apresentam uma arquitetura de um processador didático e um simulador que interage com sensores e atuadores, com o objetivo de introduzir os conceitos de IOT (Internet das Coisas) nas disciplinas de linguagem de montagem e arquitetura de computadores. O simulador pode ser executado em um computador *Raspberry Pi Nano* onde é possível ler e controlar sensores e atuadores conectados aos pinos de uso geral (GPIO) diretamente a partir do código executado pelo simulador. A abordagem é aprender de forma introdutória as questões de leitura e escrita em sensores e atuadores sem as complexidades de sistemas reais.

O trabalho de Sartor, Soares e Berejuck (2020) é focado em uma ferramenta para o estudo de arquitetura de computadores em um microprocessador simplificado chamado uPD, baseado em VHDL e com conjunto de instruções pequeno. Questionários foram aplicados aos alunos que forneceram um retorno

útil para melhora da ferramenta. Ela é utilizada em várias disciplinas do curso de engenharia de computação.

Um processador de baixa complexidade (LCP) foi criado para ser simples o bastante para ser projetado e implementado no tempo alocado às aulas de laboratório (KOSTADINOV; BENCHEVA, 2019). O processador LCP é uma máquina com arquitetura Harvard, com acumulador de 8 *bits*, memória de programa de 256x12 *bits* e memória de dados de 256 *bytes*. Duas portas de 8 *bits* fornecem comunicação com o mundo exterior e existem no conjunto mínimo de instruções há somente dois modos de endereçamento: absoluto e imediato. O montador também foi criado para rodar os programas em linguagem de montagem. Resultados indicam que os alunos tiveram motivação extra por rodar rapidamente programas no processador projetado por eles mesmos. Por outro lado, poucos estudantes declararam que tiveram uma dificuldade relativa e precisaram de ajuda.

O trabalho de Santa, Sarmiento e Ariza (2017) comenta que os conceitos que os estudantes precisam aprender sobre teoria de processadores, é obtido pelo menos em um processador de complexidade média que sempre é bem complexo para estudantes iniciantes. Para tornar mais fácil a iniciação no mundo dos processadores, é proposto um projeto com uma enorme redução de componentes de um já pequeno processador real. A abordagem minimalista torna mais fácil o entendimento do *hardware* e da linguagem de montagem, melhorando o processo de aprendizado e encorajando os estudantes a estudarem os processadores.

Esmeraldo *et al*. (2020) em seu trabalho com o simulador CompSIm também indicam que o uso de simuladores e ambientes virtuais são práticas pedagógicas complementares. Simuladores são ferramentas importantes no processo de apropriação de conhecimento e habilitam o desenvolvimento de habilidades e experiências práticas, de forma assíncrona, em cenários virtuais que lembram os reais. Simuladores também são fundamentais em laboratórios que possuem faltas de infraestrutura ou quando existe a necessidade de reduzir os custos, realizar configurações rápidas e obter resultados instantâneos. A plataforma de *hardware* do CompSim é a Mandacaru com um modelo de processador de 16 *bits* com memórias cache e RAM, barramentos de sistema e de periféricos.

O WebRISC-V é um simulador didático desenvolvido recentemente por Mariotti e Giorgi (2020) que suporta processadores e o conjunto de instruções RISC-V de 64 *bits*. O ambiente facilita para os estudantes o estudo e investigação das razões do bom desempenho de processadores pipeline, e torna possível facilmente examinar seus elementos básicos da arquitetura bem como seus estados internos. A ferramenta possui alta complexidade e não possui o objetivo de ser usada com alunos iniciantes.

O simulador LC-3, *Little Computer* 3, mimetiza a arquitetura hipotética de mesmo nome definida por Patt e Patel (2004). Ela define um banco de 8 registradores de propósito geral, 16 instruções com vários modos de endereçamento, e três *flags* Z, N e P. A chamada a sub-rotinas não é convencional e não usa explicitamente pilha, devendo ser implementada pelo programador, caso necessário.

O simulador J1 é baseado no SAP-1, Simple As Possible, (MALVINO, 1985). Ele é muito simples, com apenas 16 posições de memória e 16 instruções, incluindo

uma instrução de salto condicional. Não possui pilha, instruções para chamada de sub-rotinas e nem endereçamento indireto (MANSOUR; NAVIN; RAHMANI, 2013).

O GNUSim8085[1] é um simulador gráfico, montador e depurador para o microprocessador Intel 8085 capaz de funcionar no Linux e Windows. Este simulador suporta todas as instruções do 8085. Um código escrito em linguagem de montagem pode ser montado e, de forma transparente ao usuário, ser diretamente carregado para o simulador, permitindo a depuração diretamente do código digitado em linguagem de montagem.

O AS.SIM apresenta ideias semelhantes aos simuladores analisados. O SIMAEAC (VERONA *et al*. 2009) tem todas as 144 posições de memória do processador visíveis ao mesmo tempo. O MarieSim (NULL; LOBUR, 2003) apresenta as vantagens de utilização de uma arquitetura baseada em acumulador no aprendizado de alunos iniciantes. Percebe-se que muitos dos simuladores originais ou evoluções de simuladores compartilham características comuns, como endereçamento indireto, pilha, instruções específicas para chamada de sub-rotinas e representação de números em hexadecimal, decimal e binário. Além das características citadas, considera-se ainda a importância do aluno entender a mecânica da montagem manual de um programa e a possibilidade da montagem automática por um programa montador o qual é oferecido separado ao AS.SIM.

Os diversos trabalhos apresentados são contribuições importantes para o desenvolvimento do ensino de microprocessadores, tanto no *hardware* como no *software*. Entretanto, os requisitos de projeto de cada um deles são diferentes com poucos trabalhos voltados ao estudante iniciante no aprendizado de linguagem de montagem. O uso de simuladores de processadores RISC de 32 ou 64 *bits* tornam o processo de aprendizado mais difícil e longo. Ao mesmo tempo, as ferramentas também são voltadas a arquiteturas detalhadas com muitos elementos, o que também dificulta o processo de aprendizado. A abordagem usada neste trabalho tenta acelerar o processo de aprendizagem trabalhando com uma arquitetura simples, mas ao mesmo tempo com elementos essenciais encontrados na maioria das arquiteturas existentes.

O SIMULADOR

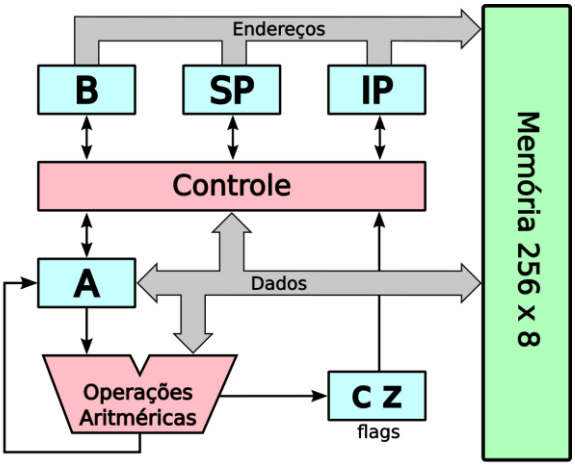
O AS.SIM foi projetado especificamente para a disciplina de Linguagem de Montagem do curso de Bacharelado de Ciências da Computação da Universidade Estadual Paulista em Bauru. Os objetivos incluem simplificar a compreensão da programação em linguagem de montagem e linguagem de máquina. Ele foi desenvolvido em *Processing*[2] pela possibilidade de ser executado em Windows, Linux e Mac. Quando se exporta uma aplicação no *Processing*, os códigos fontes acompanham a aplicação, reforçando a premissa de *software* livre.

O que fomentou o desenvolvimento do AS.SIM foi a necessidade de uma ferramenta pedagógica na forma de um simulador de uma arquitetura simplificada, que permitisse ao aluno iniciante criar um modelo mental de um computador, antes de aprofundar este aluno no aprendizado de elementos mais complexos da programação em linguagem de montagem. Neste sentido, o AS.SIM tem apenas uma janela de interface que contém todos os elementos mínimos necessários para o aprendizado dos fundamentos da programação em linguagem de montagem.

Apesar da simplicidade do AS.SIM, seu funcionamento é baseado nos mesmos conceitos fundamentais dos computadores atuais com arquitetura *von Neumann*. A arquitetura simulada pelo AS.SIM representa um computador com 16 instruções básicas, sendo que as operações aritméticas podem ser usadas com endereçamento imediato, direto e indireto por registrador. As 256 posições de memória são usadas para código, dados e pilha. A arquitetura opera com inteiros sem sinal, assim apenas os indicadores C e Z são suficientes para controlar os desvios condicionais. Apesar disso, as operações aritméticas com e sem sinal podem ser realizadas naturalmente se for considerado o complemento de dois nos valores maiores a 127.

A arquitetura conta com quatro registradores de 8 *bits*. Ela é baseada em acumulador, ou seja, um dos dados nas operações aritméticas sempre vem do acumulador, o registrador A, e o outro dado vem de um dos modos de endereçamento disponíveis. Quando a operação terminar e fornecer um resultado, ele é devolvido ao acumulador. O registrador B é destinado às operações de acesso indireto à memória, e o valor contido nele indica a posição de memória que será utilizada como um dos operandos. A arquitetura apresenta ainda dois registradores de controle, o registrador IP (*instruction pointer*) que indica a posição de memória onde está a próxima instrução a ser executada e o registrador SP (*stack pointer*) usado para manter a pilha que é base para o salvamento das posições de retorno das sub-rotinas. Os dois *flags*, ou indicadores de estados, *Carry e Zero* servem para indicar o transbordo ou zero nas operações aritméticas. A representação abstrata da arquitetura é apresentada na Figura 1.

Figura 1 – Representação da Arquitetura do AS.SIM



Fonte: Autores (2020).

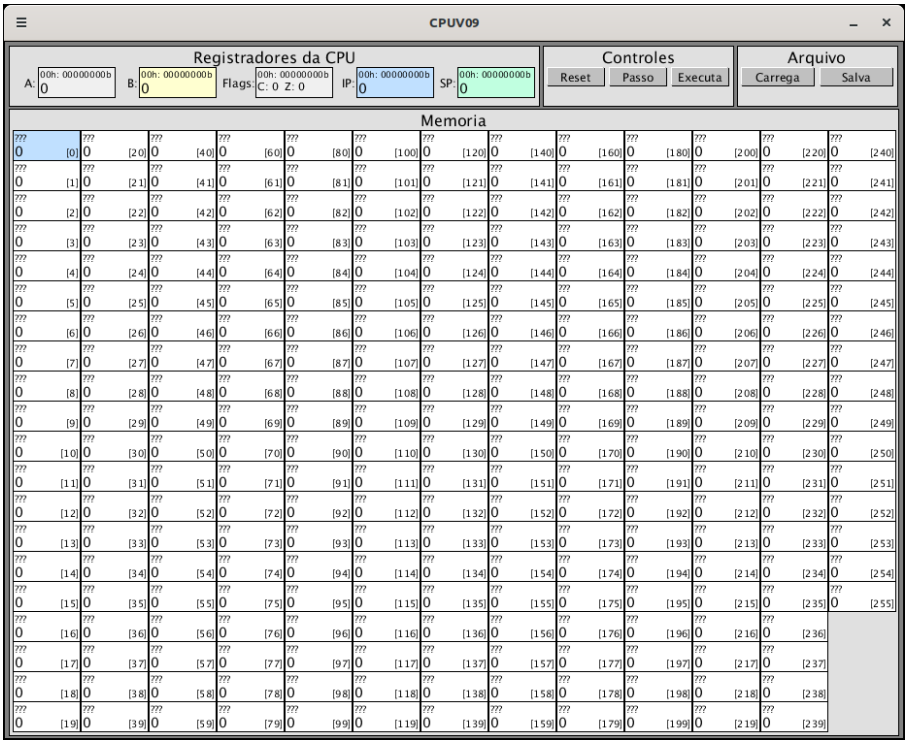
As instruções podem ser compostas por um ou dois *bytes*. O primeiro *byte*, sempre indica o *opcode* - o código da instrução a ser executada. Nas instruções de dois *bytes*, o segundo *byte* indica um argumento da instrução.

Interface

Com a intenção de construir no aluno um modelo mental de um computador, o AS.SIM apresenta todas as estruturas simuladas em uma única janela, sem menus ou configurações. Nesta janela estão representadas todas as posições de

memória e os registradores. Na mesma janela também estão dispostos 5 botões para o controle da simulação, carregamento e salvamento em arquivo dos valores armazenados na memória. Esta janela é apresentada na Figura 2.

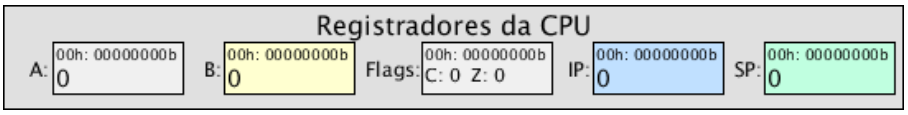
Figura 2 – Janela Única do Simulador



Fonte: Autores (2020).

Os registradores figuram no topo da interface e tem seus valores apresentados com representação decimal, hexadecimal e em binário (Figura 3). Como em um computador real, os registradores não podem ser editados diretamente pelo usuário, mas garante-se que o registrador IP e o SP são levados a zero pelo acionamento do botão ***Reset***.

Figura 3 – Registradores e *Flags* da CPU



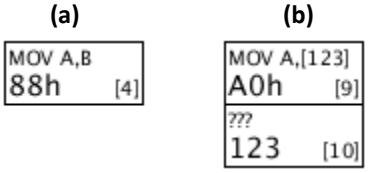
Fonte: Autores (2020).

As células da memória têm 8 *bits* e sua apresentação acontece conforme a Figura 4. Em cada célula o valor armazenado é apresentado e, caso exista uma instrução com o *opcode* correspondente, o mnemônico da instrução é indicado. No canto inferior direito existe fixo o número da posição da célula entre colchetes, o que facilita ao aluno encontrar a célula desejada e a se habituar à notação usada na sintaxe IA-32 da Intel para indicar uma posição de memória. Todas as 256 posições de memória são apresentadas ao mesmo tempo, dando uma visão global ao usuário.

A Figura 4a apresenta a célula de posição 4 da memória, que armazena o valor 88h. Nela nota-se que o *opcode* 88h corresponde à instrução “MOV A,B”, uma

instrução de apenas um *byte.* A Figura 4b apresenta as células de memória 9 e 10, com *opcode* A0h e dado 123. Observa-se que, por se tratar de uma instrução de dois *bytes*, na posição 9, a instrução apresentada é "MOV A,[123]" e o endereço 123 vem do valor do argumento fornecido na célula seguinte da memória, a célula 10.

Figura 4 – Células de memória



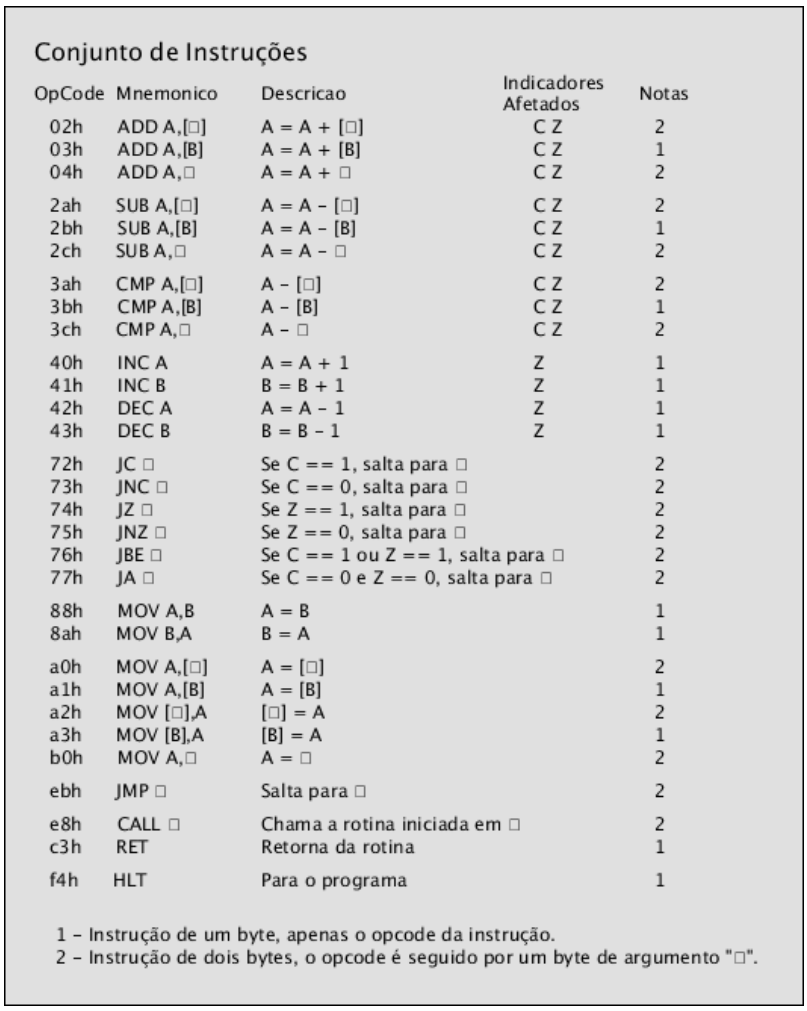
Fonte: Autores (2020).

A interface apresenta ainda 5 botões, dois voltados ao carregamento e salvamento de arquivos em linguagem de máquina e três destinados ao controle de execução do simulador. Os três botões de controle são o ***Reset*** que reinicia o processador zerando os registradores IP – contador de programa – e o SP – ponteiro da pilha –; o botão **Passo** que executa apenas a instrução indicada pelo IP, reposicionando o IP e permitindo que o usuário execute o programa instrução a instrução; e o botão **Executa** que executa sequencialmente as instruções do programa, avançando o IP e mostrando as alterações ocorridas na memória e nos registradores numa taxa de duas instruções por segundo. Durante a execução do programa, iniciada pelo acionamento do botão **Executa**, o rótulo deste botão passa a ser **Para**, indicando a possibilidade de parada do programa na posição atual da execução.

Tentando manter o mais simples possível a interface com o usuário, o AS.SIM conta com a janela principal e apenas uma outra de ajuda indicando informações úteis sobre as instruções disponíveis no simulador (Figura 5). A janela de ajuda do AS.SIM estimula o uso da sintaxe de instruções da Intel utilizadas pelos montadores para a IA-32 o que facilita, num segundo momento na disciplina, que o aluno programe um computador real sem o impacto do aprendizado de novos mnemônicos para a nova arquitetura.

A janela de ajuda apresenta uma tabela onde cada linha corresponde a uma instrução e as colunas trazem as seguintes informações: i) *opcode* da instrução; ii) mnemônico; iii) uma descrição resumida do funcionamento no formato semelhante ao usado em linguagens de alto nível; iv) os indicadores *flags* afetados pela instrução; e v) uma nota sobre o número de *bytes* da instrução. O símbolo "□" representa onde será inserido o valor do segundo *byte* que compõe a instrução. Os colchetes, "[" e "]", representam o endereçamento direto ou indireto com registradores. A célula de memória na posição indicada entre eles será usada pela instrução.

Figura 5 – Janela de ajuda do AS.SIM



Conjunto de Instruções

OpCode	Mnemonico	Descricao	Indicadores Afetados	Notas
02h	ADD A,[□]	A = A + [□]	C Z	2
03h	ADD A,[B]	A = A + [B]	C Z	1
04h	ADD A,□	A = A + □	C Z	2
2ah	SUB A,[□]	A = A - [□]	C Z	2
2bh	SUB A,[B]	A = A - [B]	C Z	1
2ch	SUB A,□	A = A - □	C Z	2
3ah	CMP A,[□]	A - [□]	C Z	2
3bh	CMP A,[B]	A - [B]	C Z	1
3ch	CMP A,□	A - □	C Z	2
40h	INC A	A = A + 1	Z	1
41h	INC B	B = B + 1	Z	1
42h	DEC A	A = A - 1	Z	1
43h	DEC B	B = B - 1	Z	1
72h	JC □	Se C == 1, salta para □		2
73h	JNC □	Se C == 0, salta para □		2
74h	JZ □	Se Z == 1, salta para □		2
75h	JNZ □	Se Z == 0, salta para □		2
76h	JBE □	Se C == 1 ou Z == 1, salta para □		2
77h	JA □	Se C == 0 e Z == 0, salta para □		2
88h	MOV A,B	A = B		1
8ah	MOV B,A	B = A		1
a0h	MOV A,[□]	A = [□]		2
a1h	MOV A,[B]	A = [B]		1
a2h	MOV [□],A	[□] = A		2
a3h	MOV [B],A	[B] = A		1
b0h	MOV A,□	A = □		2
ebh	JMP □	Salta para □		2
e8h	CALL □	Chama a rotina iniciada em □		2
c3h	RET	Retorna da rotina		1
f4h	HLT	Para o programa		1

1 - Instrução de um byte, apenas o opcode da instrução.
2 - Instrução de dois bytes, o opcode é seguido por um byte de argumento "□".

Fonte: Autores (2020).

Funcionamento

As simulações no AS.SIM acontecem a partir das instruções carregadas na memória, o que pode ser feito diretamente pelo usuário, digitando os *opcodes* das instruções e seus dados nas células de memória, ou pela carga de um arquivo objeto com os dados a serem atribuídos à memória.

Para que o usuário possa inserir um programa nas células de memória, ele precisa entender os princípios da montagem, ou seja, a transformação de linguagem de montagem para linguagem de máquina. Assim, ele escreve seu programa em linguagem de montagem usando os mnemônicos e os dados de cada instrução, e em seguida faz a montagem manualmente, que consiste na transformação dos mnemônicos para os *bytes* dos *opcodes* correspondentes, gerando uma sequência dos *opcodes* e dados. Então ele insere esta sequência, *byte* e *byte*, nas células de memória do simulador. A montagem deve respeitar as instruções disponíveis, apresentadas na janela de ajuda, conforme a Figura 5.

O arquivo objeto usado pelo AS.SIM é um arquivo texto contendo até 256 linhas, sendo que cada linha corresponde à célula de memória correspondente. Este arquivo pode ser editado pelo usuário em um editor de texto ou gerado por

um programa montador a partir de um arquivo fonte em linguagem de montagem. Um programa montador automatiza a transformação de um arquivo em linguagem de montagem para um arquivo em linguagem de máquina, sendo este segundo chamado de arquivo objeto. Tanto no arquivo objeto quanto nos *opcodes* e dados digitados, cada valor pode estar representado em decimal ou hexadecimal quando seguido por um “h”.

O AS.SIM opera com saltos e chamadas às sub-rotinas com destinos absolutos, pois são mais intuitivos e facilitam o cálculo dos destinos na montagem. As instruções de saltos condicionais ou fixos, quando no momento dos seus desvios, fazem com que o segundo *byte* da instrução, o endereço do destino, seja carregado no IP. De forma semelhante, quando uma instrução CALL é usada, o IP+2 é salvo na pilha e IP é carregado com o endereço da sub-rotina. Na instrução RET atribui-se ao IP o valor retirado da pilha, retornando da sub-rotina.

Na IA-32, as instruções de saltos condicionais são divididas em dois grupos: saltos condicionais com e sem sinal, sendo que as operações com sinal usam além dos indicadores C e Z, também indicadores de sinal e estouro (INTEL, 2016). No AS.SIM, uma vez que existem apenas os indicadores C e Z, as instruções de saltos condicionais sempre operam a partir dos indicadores afetados nas instruções aritméticas sobre inteiros sem sinal e acompanham a nomenclatura dos mnemônicos da IA-32 para os desvios condicionais aplicados a inteiros sem sinal. As seis instruções de saltos condicionais, quando empregadas com a instrução SUB ou CMP, contemplam as condições de menor (JC), maior (JA), igual (JZ), diferente (JNZ), menor ou igual (JBE) e maior ou igual (JNC). As instruções JC, JNC, JZ, JNZ verificam apenas um indicador para decidir pelo desvio, as instruções JA e JBE dependem dos dois indicadores, sendo que JA verifica de C e Z estão desligados e JBE verifica se C ou Z estão ligados.

O USO DO AS.SIM NA DISCIPLINA DE LINGUAGEM DE MONTAGEM

O AS.SIM tem sido usado desde 2014 na disciplina de Linguagem de Montagem do curso de Ciência da Computação da Universidade Estadual Paulista – UNESP de Bauru. Esta disciplina semestral tem 2 horas semanais e figura no segundo semestre do curso. O AS.SIM é uma ferramenta introdutória, empregada nas primeiras 12 horas desta disciplina.

Um ambiente integrado para desenvolvimento de programas profissionais pode esconder passos ou gerar compreensões erradas de como um programa é montado e carregado na memória. Diante disso, com o uso do AS.SIM, o aluno é incentivado a escrever o programa inicialmente em linguagem de montagem sem nenhuma diretiva, e em seguida realizar a montagem manualmente e testar seus programas. Como os alunos inicialmente precisam escrever o programa diretamente em linguagem de montagem, as relações entre linguagens de máquina, montagem e de alto nível são reforçadas, o que aprofunda os conceitos de fundamentos de computação.

Uma vez que o aluno já tenha adquirido o conceito da montagem, são inseridos as diretivas e rótulos para valores constantes e de posições de memória, para que então o aluno passe a usar um programa montador. Paralelamente, o aluno é incentivado a desenvolver seu próprio programa montador, com orientação do professor sobre as especificações do montador, as diretivas,

instruções, modos de endereçamento e no formato normalmente encontrado nos montadores comerciais. Ele instrui também que o montador precisa realizar dois passos para completar a montagem: o primeiro passo para determinar a posição dos rótulos usados nos destinos aos desvios e às sub-rotinas, e o segundo passo com a montagem das instruções e gravação da linguagem de máquina no arquivo objeto. O montador proposto, além das instruções disponíveis e indicadas na janela de ajuda (Figura 5), oferece as diretivas ORG, EQU, DB e END.

Uma vez que o estudante já tenha entendido o funcionamento da arquitetura de uma CPU, sua programação em linguagem de máquina e de montagem, o aluno é levado a outra ferramenta, com ambiente de desenvolvimento de programas profissionais, mas que em seu ambiente integrado ofereça compilação, montagem, carga, execução e depuração do código com a visualização dos registradores, seções de memória e *flags*.

RESULTADOS

O AS.SIM tem sido usado desde 2014 no Curso de Ciências da Computação da Faculdade de Ciências da Unesp de Bauru, o que permite uma análise quantitativa baseada no número de aprovações na disciplina de Linguagem de Montagem.

Foram levantados dados de aprovação dos alunos que cursaram a disciplina de Linguagem de Montagem entre 2011 e 2018, neste período, ingressaram 325 alunos, incluindo os que desistiram e os que reprovaram e refizeram no ano seguinte. As proporções de aprovados são apresentadas na Tabela 1. O AS.SIM foi utilizado pelos alunos nos anos 2014, 2015, 2016 e 2018. Em 2017, um professor substituto ministrou a disciplina e não usou o AS.SIM.

Tabela 1 – Alunos Aprovados em Cada Ano

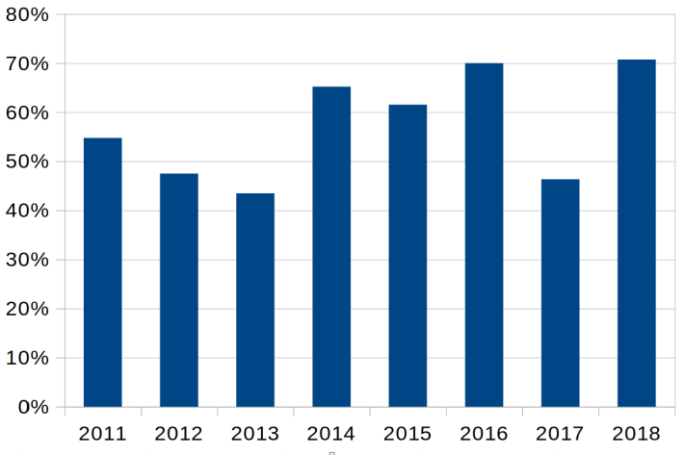
Ano	Total de alunos	Aprovados	Aprovados (%)
2011	42	23	55%
2012	40	19	48%
2013	46	20	43%
2014	46	30	65%
2015	39	24	62%
2016	30	21	70%
2017	41	19	46%
2018	41	29	71%

Fonte: Autores (2020).

Analisando o gráfico de porcentagens de aprovação dos alunos na disciplina, apresentado na Figura 6, o qual foi construído a partir da última linha da Tabela 1, observa-se que nos anos 2014, 2015, 2016 e 2018, quando o simulador foi usado, houve uma porcentagem de aprovação maior ou igual a 62%. Nos outros anos, a aprovação foi inferior a 55%. Ignorando outros fatores adversos que podem ter

influenciado a taxa de aprovação, parece significativa a melhora apresentada pelos alunos que usaram o AS.SIM.

Figura 6 – Porcentagem de Aprovados por Ano



Fonte: Autores (2020).

A Tabela 2 apresenta a distribuição do número de alunos da disciplina divididos por faixas de notas finais, sendo que cada faixa tem largura de 1,0 e que as notas finais atribuídas aos alunos vão de 0,0 a 10,0. Os alunos que desistiram estão na faixa de [0, 1]. Nesta Tabela, os alunos estão divididos nas turmas que utilizaram o simulador, com 156 alunos; e nas turmas que não usaram o simulador, com 169 alunos.

Tabela 2 – Distribuição das Notas por Faixas

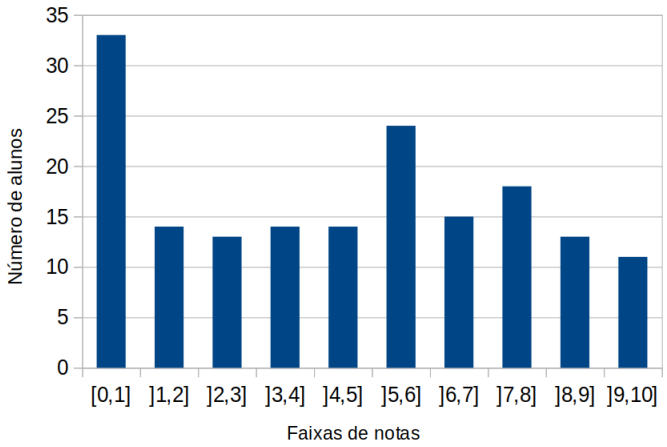
Notas	Com simulador	Sem simulador
[0, 1]	2	17
]1, 2]	7	13
]2, 3]	16	14
]3, 4]	5	13
]4, 5]	13	23
]5, 6]	22	15
]6, 7]	26	16
]7, 8]	19	19
]8, 9]	20	14
]9,10]	14	8

Fonte: Autores (2020).

Comparando as representações gráficas na Figura 7 e na Figura 8 da Tabela 2, observa-se que a faixa de alunos com notas [0, 1], que não usaram o simulador (Figura 7), é preponderante. Por outro lado, nos alunos que usaram o AS.SIM (Figura 8) a maior concentração de notas está na faixa [6, 7]. Esta observação

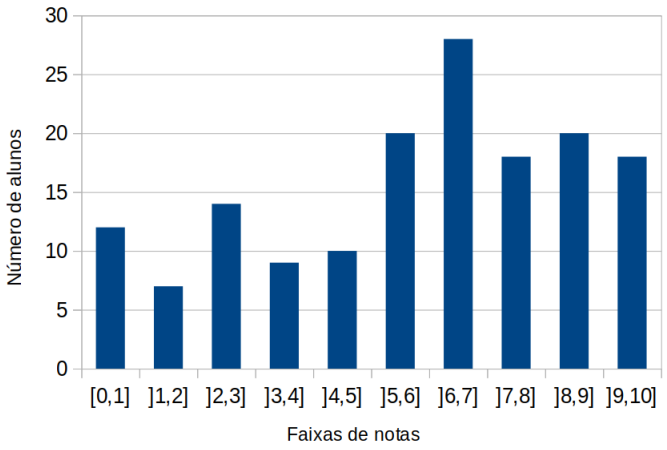
evidencia que o uso de uma ferramenta pedagógica bem adaptada ao domínio lecionado pode incentivar o aluno, diminuindo a evasão e melhorando o seu aproveitamento na disciplina.

Figura 7 – Distribuição das Notas dos Alunos nas Turmas Sem o uso do AS.SIM



Fonte: Autores (2020).

Figura 8 – Distribuição das Notas dos Alunos nas Turmas Com o uso do AS.SIM



Fonte: Autores (2020).

Os alunos relatam que a execução direta do simulador sem a necessidade de instalação, a sua interface fácil de usar e, principalmente, a simplicidade na arquitetura da CPU simulada os ajudaram no entendimento dos conceitos de linguagem de montagem.

CONCLUSÕES

O AS.SIM é uma proposta de um simulador simples baseado no modelo de von *Neumann* para iniciar o ensino de Linguagem de Montagem. Para o professor, é uma opção de uma ferramenta pedagógica entre outros simuladores com objetivos semelhantes. A escolha pelo AS.SIM pode ser justificada pela facilidade de explicar as funcionalidades simples deste simulador, por ser fácil de usar e de instalar na maioria dos sistemas operacionais comuns utilizados em computadores pessoais.

Pela simplicidade do AS.SIM, o aluno coloca seus esforços no aprendizado da linguagem de montagem e de sua conversão para linguagem de máquina. A utilização do simulador não demanda esforços extras do aluno, pois consiste apenas em carregar os *opcodes* e dados na memória e acionar o botão de **Executa** para testar o seu programa.

A partir dos dados obtidos no período de 2011 a 2018 no ensino da disciplina de Linguagem de Montagem na Unesp de Bauru, nota-se indicações de vantagens na utilização deste simulador. Estes dados evidenciam que o uso de uma ferramenta pedagógica, bem adaptada ao domínio lecionado, pode melhorar o aproveitamento dos alunos.

A SIMULATOR TO SUPPORT ASSEMBLY LANGUAGE TEACHING

ABSTRACT

This article describes a simulator of a simple hypothetical processor used for introducing assembly language concepts to high school and university students. The simulator, developed to be used as a didactic tool, offers to students, in a graphical interface, a model of how a computer works from the point of view of low-level programming. In this tool, users load the program in machine language and visualize the changes resulting from its execution in the processor's memory and registers. Considering the importance of understanding the difference between assembly language and machine language, students are instructed to write their code in assembly language and then to obtain the machine language. The assembly process begins manually and then it is done through an assembler program. Manual assembly helps to explain some concepts related to the generation of executable code hidden in integrated development environments. Although the tool simulates a simple hypothetical processor, it was built following the instruction syntax used in Intel's 32-bit architecture (IA-32), allowing students to use the concepts learned to understand other assembly languages on real computers. This tool is used at the beginning of the Assembly Language course in the Computer Science Program at Sao Paulo State University, located in Bauru/SP/BR. Data collected over eight years, four of them using the simulator, suggest the pass rate of students has increased significantly.

KEYWORDS: Assembly Language. Simulator. Computing Teaching.

NOTAS

1 O GNUSIM8085 foi escrito por Sridhar Ratnakumar em 2003. Disponível em: http://gnusim8085.srid.ca/. Acesso em: 10 dez. 2019.

2 O *Processing*. Disponível em: https://processing.org/. Acesso em: 15 mai. 2019.

REFERÊNCIAS

ACM/IEEE. Joint Task Force on Computing Curricula. **Association for Computing Machinery (ACM) and IEEE Computer Society**: Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. ACM and IEEE Computer Society, 20 de dez. de 2013. Disponível em: https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf. Acesso em: 21 abr. 2020.

ACM/IEEE. Joint Task Force on Computing Curricula. **Association for Computing Machinery (ACM) and IEEE Computer Society**: Computer Engineering Curricula 2016: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering. ACM and IEEE Computer Society, 15 de dez. de 2016. Disponível em: https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf. Acesso em: 21 abr. 2020.

ESMERALDO, G.; LISBOA, E. B. Uma Ferramenta para Exploração do Ensino de Organização e Arquitetura de Computadores. **International Journal of Computer Architecture Education**, v. 6. p. 68-75, 2017.

ESMERALDO, G. et al. Um Estudo Comparativo entre Simuladores Computacionais para Apoio à Disciplina de Arquitetura e Organização de Computadores. In: CONGRESSO SOBRE TECNOLOGIAS NA EDUCAÇÃO (CTRL+E), 4., 2019, Recife. **Anais do IV Congresso sobre Tecnologias na Educação**. Porto Alegre: Sociedade Brasileira de Computação, p. 434-443, 2019.

ESMERALDO, G. Á. et al. CompSim: An Integrated Environment for Learning and Designing of Embedded Computational Systems. In: 2020 **XIV Technologies Applied to Electronics Teaching Conference (TAEE). IEEE**. p. 1-4, 2020.

INTEL**. Intel® 64 and IA-32 Architectures Software Developer's Manual**. Volume 1: Basic Architecture. p. 482, 2016. Disponível em: https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-1-manual.html. Acesso em: 21 ago. 2018.

KOSTADINOV, N.; BENCHEVA, N. An Approach for Teaching Processor Design and How to Extend its Features. In: 2019. **29th Annual Conference of the European**

Association for Education in Electrical and Information Engineering (EAEEIE). IEEE, p. 1-4, 2019.

MALVINO, A.P. **Microcomputadores e Microprocessadores**. McGRAW-HILL do Brasil, 578 p., 1985.

MANSOUR, A. Y.; NAVIN, A. H.; RAHMANI, A. M. J1 Accumulator-Based Processor for Educational Purposes. **International Journal of Advanced Research in Computer Science**, v. 4 n. 4, p. 191-194, mar./.abr. 2013.

MARIOTTI, G.; GIORGI, R. **WebRISC-V**: A RISC-V Educational Simulator featuring RV64IM, Pipeline and Web-Based UI. Department of Information Engineering and Mathematics, University of Siena, July 2020.

NIKOLIC, B. *et al*. A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization, **IEEE Transactions on Education**, v. 52, n. 4, p. 449-458, nov. 2009.

NULL, L.; LOBUR, J. MarieSim: The MARIE Computer Simulator. **ACM Journal on Educational Resources in Computing (JERIC)**, v. 3, n. 2, p. 1-29, 2003.

PATT, Y. N.; PATEL, S. **Introduction to Computing Systems**: From Bits and Gates to C and Beyond. 2. ed. New York, NY: McGraw-Hill Higher Education. 632 p., 2004.

SANTA, F. M.; SARMIENTO, F. H. M.; ARIZA, H. M. Minimalist 4-bit Processor Focused on Processors Theory Teaching. **Indian Journal of Science and Technology**, v. 10, n. 14, p. 1-6, 2017.

SARTOR, M.; SOARES, T. T. M. S.; BEREJUCK, M. D. Building a microprocessor architecture at Computer Engineering undergraduate courses. Building a microprocessor architecture at Computer Engineering undergraduate courses. **International Journal of Advanced Engineering Research and Science**, v. 7, n. 7. Jul. 2020.

SILVA, G. P.; BORGES, J. A. S. SimuS - Um Simulador Para o Ensino de Arquitetura de Computadores. **International Journal of Computer Architecture Education (IJCAE)** v. 5, n. 1, p. 7-12, 2016.

SILVA, G. P.; BORGES, J. A. S. A Didactic Processor and Simulator for IoT. In: 2018 3rd International Conference of the Portuguese Society for Engineering Education (CISPEE), Aveiro. 2018, **3rd International Conference of the Portuguese Society for Engineering Education (CISPEE)**, 2018. v. 1, p. 1-7, 2018.

VERONA, A. B.; MARTINI, J. A.; GONÇALVES, T. L. SIMAEAC: Um Simulador Acadêmico para Ensino de Arquitetura de Computadores. **Varia Scientia**, v. 9, n. 16, p. 139-148, 2009.

Recebido: 15 ago. 2020.
Aprovado: 23 nov. 2021.
DOI: 10.3895/rbect.v15n2.13014
Como citar: PEGORARO, R.; FRANCHIN, M. N. Um simulador de apoio ao ensino de linguagem de montagem. **Revista Brasileira de Ensino de Ciência e Tecnologia**, Ponta Grossa, v.15, p. 1-19, 2022. Disponível em: <https://periodicos.utfpr.edu.br/rbect/article/view/13014>. Acesso em: XXX.
Correspondência: Rene Pegoraro - rene.pegoraro@unesp.br