colloquium exactarum

SIMULADOR DE UCP COM SUPORTE À MEMÓRIA CACHE E PIPELINE

UCP SIMULATOR WITH CACHE MEMORY AND PIPELINE

Artur Jordão Lima Correia; Mario Augusto Pazoti; Francisco Assis da Silva; Leandro Luiz de Almeida; Danillo Roberto Pereira

Universidade do Oeste Paulista – UNOESTE, Faculdade de Informática – FIPP, Presidente Prudente – SP. E-mail: arturjordao@unoeste.edu.br, {mario, chico, llalmeida, danilopereira}@unoeste.br

RESUMO – Um problema comum em disciplinas de arquitetura de computadores é a dinâmica real dos processos que ocorrem internamente em hardware. O funcionamento de uma UCP, por exemplo, é algo complexo e sua compreensão é fundamental para utilização de seus recursos. Este trabalho contribui com o desenvolvimento de um simulador de UCP com suporte à memória *cache* e *pipeline*. São apresentados os resultados obtidos com a ferramenta desenvolvida, e que a utilização do simulador pode ser muito útil como apoio em disciplinas de arquitetura de computadores, melhorando o entendimento por parte dos alunos.
Palavras-chave: Simulador UCP; Cache; *Pipeline*.

ABSTRACT – A common problem in computer architecture disciplines is the real dynamics of the processes occurring internally in hardware. The working of a CPU, for example, is complex and its understanding is fundamental to the use of their resources. This work contributes to the development of a UCP simulator with support to cache and pipeline. We presented the results obtained with the developed tool, and that the use of the simulator can be very useful to support disciplines of computer architecture, improving the understanding by the students.
Keywords: CPU Simulator; Cache; Pipeline.

Recebido em: 20/06/2014
Revisado em: 10/08/2014
Aprovado em: 15/09/2014_

1 INTRODUÇÃO

Ao longo das gerações de processadores, algumas características básicas têm se mantido, como por exemplo, algumas instruções, cache de memória, *pipeline* etc. Diversas literaturas apresentam um diagrama de uma UCP simples, utilizada para fins didáticos em disciplinas de arquitetura de computadores que contemplam tais características. A compreensão do funcionamento dessa arquitetura básica, entretanto, é por vezes difícil ao aluno e a complexidade em se analisar o funcionamento de uma UCP em hardware torna o uso de um simulador uma opção desejável. Acredita-se que o uso de simuladores possibilita melhor entendimento do funcionamento da UCP.

Algumas tecnologias para aumentar o desempenho da UCP como, por exemplo, o uso de memória *cache* e *pipeline* também passaram a ser conceitos básicos em uma arquitetura de computador.

Este trabalho visa integrar um único simulador uma arquitetura de processador contendo os recursos de memória cache e *pipeline*, a fim de apresentar o comportamento de cada microinstrução e a comunicação entre os componentes da UCP e memória RAM.

O presente artigo está organizado em seis seções, sendo que na Seção 2 são apresentados os componentes da UCP, suas instruções e tecnologias utilizadas para aumentar seu desempenho. A Seção 3 apresenta os trabalhos semelhantes, levantando suas características, a Seção 4 trata sobre o desenvolvimento do projeto, mostrando o software que foi desenvolvido e as ferramentas utilizadas. Por fim, a Seção 5 e a Seção 6 mostram respectivamente os resultados obtidos com a ferramenta e a conclusão do trabalho.

2 ARQUITETURA DA UCP

Uma UCP simples é basicamente composta por registradores e barramentos. Na arquitetura proposta por Taub (1984) os registradores que compõe a UCP são: Contador de programa (PC), registrador de endereço de memória (MAR), registrador de operação (OPR), registrador de propósitos gerais (GPR), registrador do acumulador (ACC) e um *flip-flop* (F). A UCP contém ainda um somador e uma unidade de controle (UC). Na Figura 1 são mostrados os componentes citados e as microoperações que eles podem realizar.

O tamanho em bits de cada registrador influencia diretamente no valor máximo do número que é possível trabalhar, seja no registrador utilizado em operações de lógica e aritmética, ou na quantidade de memória que se pode endereçar.

Na arquitetura adotada, o contador de programa e o registrador de memória possuem o tamanho de oito bits e são responsáveis, respectivamente, em armazenar o endereço da próxima instrução e o endereço da memória que será endereçada. O dado que trafega entre a memória e a UCP é armazenado no registrador de propósitos gerais que tem 12 bits de tamanho.

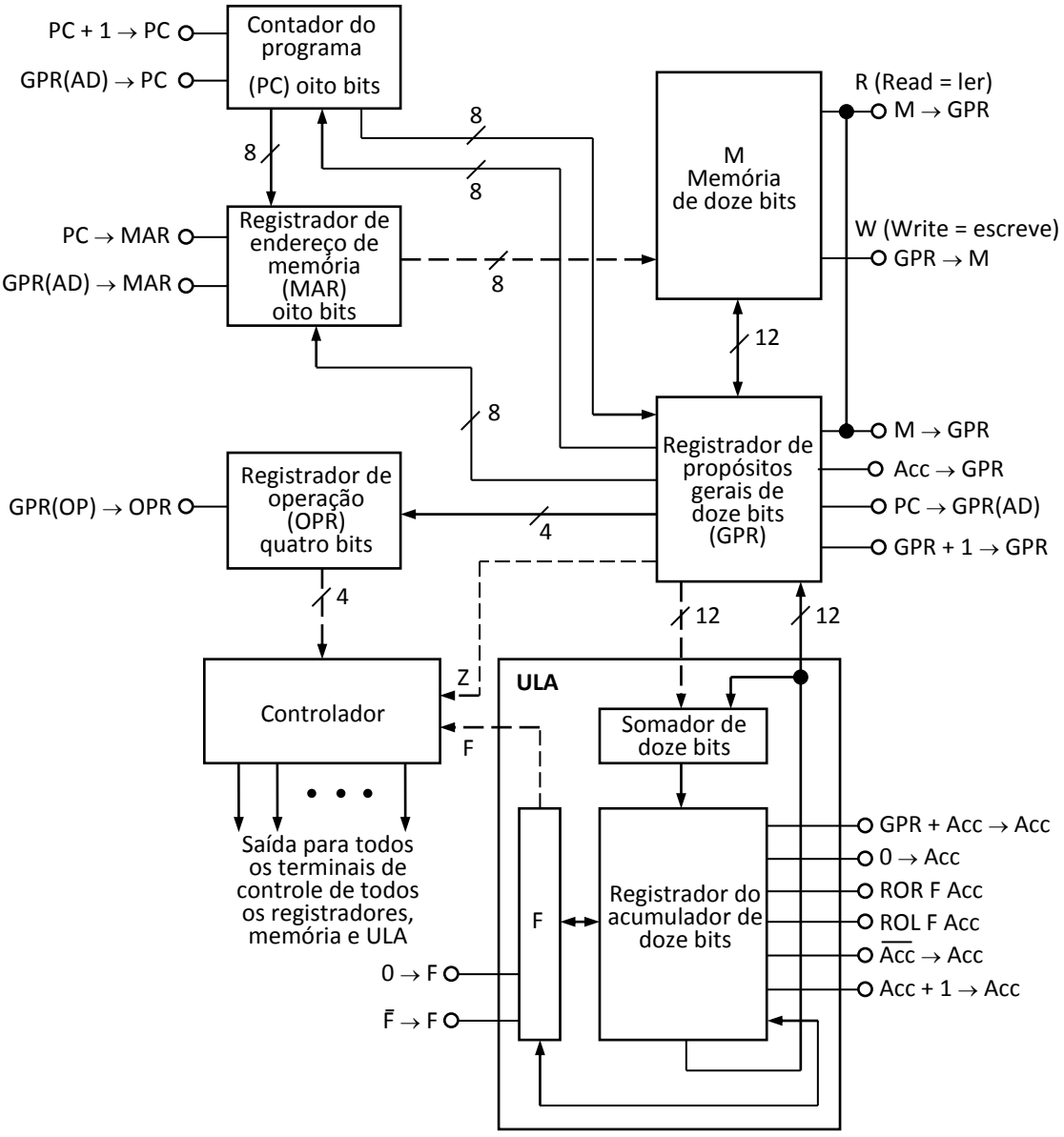


Figura 1. Arquitetura simples de um computador.
Fonte: (TAUB, 1984).

Para realizar as operações lógicas e aritméticas, a UCP faz uso da ULA (Unidade Lógica e Aritmética), que é composta por um registrador acumulador de 12 bits, um somador de 12 bits e um *flip-flop* F (de um bit) que é utilizado em instruções de rotação. A instrução a ser executada é armazenada no registrador de operações de quatro bits,

sendo que o tamanho desse registrador influencia na quantidade de instruções possíveis para a arquitetura adotada. No caso de quatro bits, o repertório é de 16 instruções.

Os 12 bits do GPR são divididos em duas partes, conforme mostrado na Figura 2, os quatro bits mais significativos (Opcode), os da esquerda, representam a instrução que foi lida da memória e será enviada para o OPR e posteriormente executada. Os bits restantes (Operando) são utilizados para representar o endereço de memória que será transferido para o MAR. A capacidade máxima do GPR será usada quando ele estiver retendo um número e não uma instrução.

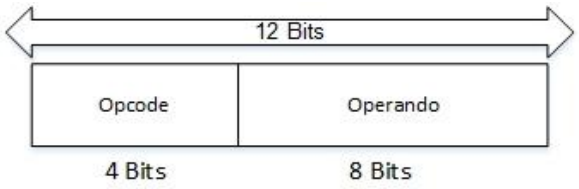


Figura 2. Formato da instrução.

2.1 Instrução de Máquina

Com um OPR de 12 bits é possível um repertório de 16 microinstruções (Tabela 1).

As instruções mostradas no Tabela 1 são propostas por Taub (1984). Cada uma dessas microinstruções é dividida em uma ou mais microoperações. Na Figura 1 são mostradas quais microoperações cada componente da UCP pode realizar.

Tabela 1. Repertório de instruções.

Instrução	Função
CRA	Limpar o acumulador
CTA	Complementar o acumulador
ITA	Incrementar o acumulador
CRF	Limpar o flip-flop F
CTF	Complementar o flip-flop F
SFZ	Saltar próxima instrução se F=0
ROR	Girar à direita
ROL	Girar à esquerda
ADD	Adicionar ao acumulador
ADDI	Adicionar ao acumulador, indireto
STA	Armazenar na memória do acumulador
JMP	Saltar
JMPI	Saltar, indireto
CSR	Chamar sub-rotina

Fonte: (TAUB, 1984).

Para que uma microinstrução seja executada na arquitetura de Von Neumann, há um ciclo de instrução dividido em duas etapas: busca e execução. A busca consiste em trazer a instrução da memória principal para dentro da UCP e apontar para a próxima instrução que será executada. A execução consiste na realização da operação propriamente dita (STALLINGS, 2003).

2.2 Tecnologias para Aumentar o Desempenho da Máquina

O aumento da velocidade dos processadores não tem sido acompanhado pelo aumento da velocidade da memória principal (TANEMBAUM, 2007). Assim, a utilização de memória *cache* se fez necessária para minimizar esse problema. A memória *cache* está fisicamente localizada próxima ao processador e armazena uma pequena cópia

da memória principal. Quando o processador necessita buscar uma instrução, ele a busca inicialmente na memória *cache*, caso ele encontre (*cache hit*) a instrução é levada para dentro da UCP, caso contrário (*cache miss*) a instrução deve ser colocada na *cache* e solicitada novamente. A memória *cache* é dividida em blocos de tamanhos fixos denominados linhas de *cache*, quando ocorre um *cache miss* toda a linha é substituída (TANEMBAUM, 2007).

De acordo com Tanembaum (2007) as operações de *cache* são baseadas em dois princípios: princípio da localidade espacial e temporal. A localidade espacial estabelece que os endereços de memória que estão próximos ao endereço que foi acessado recentemente têm grande possibilidade de serem acessados também. Já o princípio da localidade temporal propõe que os endereços que foram visitados recentemente voltarão a ser utilizados futuramente (TANEMBAUM, 2007).

Quando uma linha de *cache* precisa ser substituída, a escolha pode ser feita de várias formas. Os dois algoritmos mais comuns de substituição de blocos são: LRU (*Last Recently Used* – Não usado há mais tempo) e FIFO (*First Input First Output* – Primeiro a chegar primeiro a sair). O LRU é o mais eficaz, sendo que o bloco escolhido para ser substituído é o que está há mais tempo sem ser utilizado (STALLINGS, 2003). Por outro lado, a política do FIFO consiste na escolha do bloco que está há mais tempo na memória *cache* (STALLINGS, 2003).

Quando o processador altera um dado na *cache*, o dado deve ser atualizado na memória principal, existem duas formas de realizar essa atualização: Write-Through e Write-Back. A primeira técnica é a mais simples e propõe que a memória principal seja atualizada no mesmo momento em que o dado é alterado na memória *cache*. A segunda atualiza a memória principal somente quando a linha em que o dado está é removida da *cache*. Um dos problemas na utilização deste método é que partes da memória principal podem ficar inválidas (com valores obsoletos) (STALLINGS, 2003).

Outra forma de melhorar o desempenho da máquina é a utilização da *pipeline* de instruções. Essa técnica faz com que novas instruções sejam aceitas pela UCP antes que as instruções em execução tenham sido finalizadas. Para utilização da *pipeline*, uma instrução deve ser dividida em estágios que ocorrem em sequência. Com isso permite-se que instruções sejam executadas em paralelo desde que estejam em estágios diferentes (STALLINGS, 2003). Nos processadores atuais são utilizados dez ou mais estágios (TANEMBAUM, 2007).

Utilizando a técnica da *pipeline*, a execução de instruções é acelerada, pois se o estágio de busca e execução tiver a mesma

duração, assim o número de instruções finalizadas em um intervalo de tempo é dobrado (STALLINGS, 2003).

No caso da *pipeline*, uma das desvantagens está no processamento de instruções de desvio, as quais podem invalidar a execução de outro estágio já carregado na UCP. Esse fenômeno é denominado esvaziamento da *pipeline*. Na Figura 3 é mostrado um pseudocódigo que representa o esvaziamento da *pipeline*, para N!=0, visto que a busca realizada para a instrução 2 não será aproveitada, já que haverá um desvio ocasionada na execução da Instrução 1 para a Instrução 3.

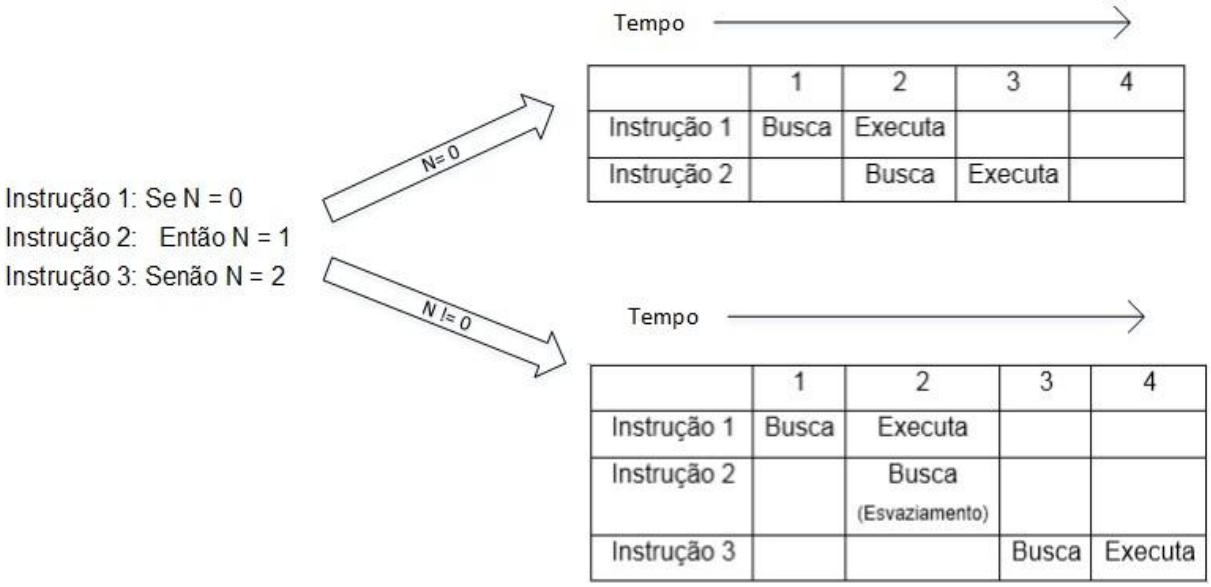


Figura 3. Pseudocódigo de esvaziamento da *pipeline*.

Outro problema envolvendo a utilização da *pipeline* é pelo fato da disputa pela utilização dos mesmos componentes da UCP em certos momentos, fazendo que a nova instrução entre em espera (estado ocioso). A diferença no tempo entre os estágios também pode fazer a instrução entrar no estado de espera.

3 TRABALHOS RELACIONADOS

Investigou-se na literatura alguns trabalhos correlatos ao apresentado neste artigo, sendo eles: o UCP-8E (SILVA-FILHO, 2013), Sistema Simulador de Arquitetura Básica de Processadores (SANTOS, 2005) e Simulador de uma Arquitetura Básica de Computadores (SITOLINO, 1992), desenvolvidos seguindo como base a arquitetura de Von Neumann, o objetivo de todos esses simuladores é conseguir de

maneira fácil e simples acompanhar as fases de execução de uma instrução.

A ferramenta desenvolvida por Sitolino (1992) apresenta um conjunto de 16 instruções propostas por Taub (1984), permitindo substituir instruções ou alterar o funcionamento das instruções existentes. Foi desenvolvida na linguagem C, com interface texto. Para a criação do programa, as instruções são selecionadas por meio de um código.

O Simulador de Arquitetura Básica de Processadores segue também o conjunto de 16 instruções (TAUB, 1984). Desenvolvido em Pascal, o simulador possui interface gráfica permitindo um melhor acompanhamento do processo de execução da instrução.

Diferente das outras duas ferramentas citadas, o UCP-8E (UCP educacional de oito bits) possui um conjunto de instruções próprio desenvolvido pelos autores do projeto. As instruções são colocadas por meio de comandos em Hexadecimal.

4 DESENVOLVIMENTO DO PROJETO

Para o desenvolvimento do projeto foi utilizada a arquitetura e instruções propostas por Taub (1984), seguindo as definições de uma arquitetura de Von Neumann.

Na Figura 4 é possível visualizar o simulador com suas funcionalidades numeradas por meio de marcadores que representam:

1. Diagrama de UCP com seus registradores e barramentos;
2. Memória principal;
3. Memória *cache*;
4. Configurações que o simulador permite;
5. Configurações adicionais da memória *cache*;
6. Editor de instruções;
7. Histórico de instruções e *pipeline*;
8. Iniciar o programa inserido;
9. Para o programa;
10. Passar para próxima etapa da execução.

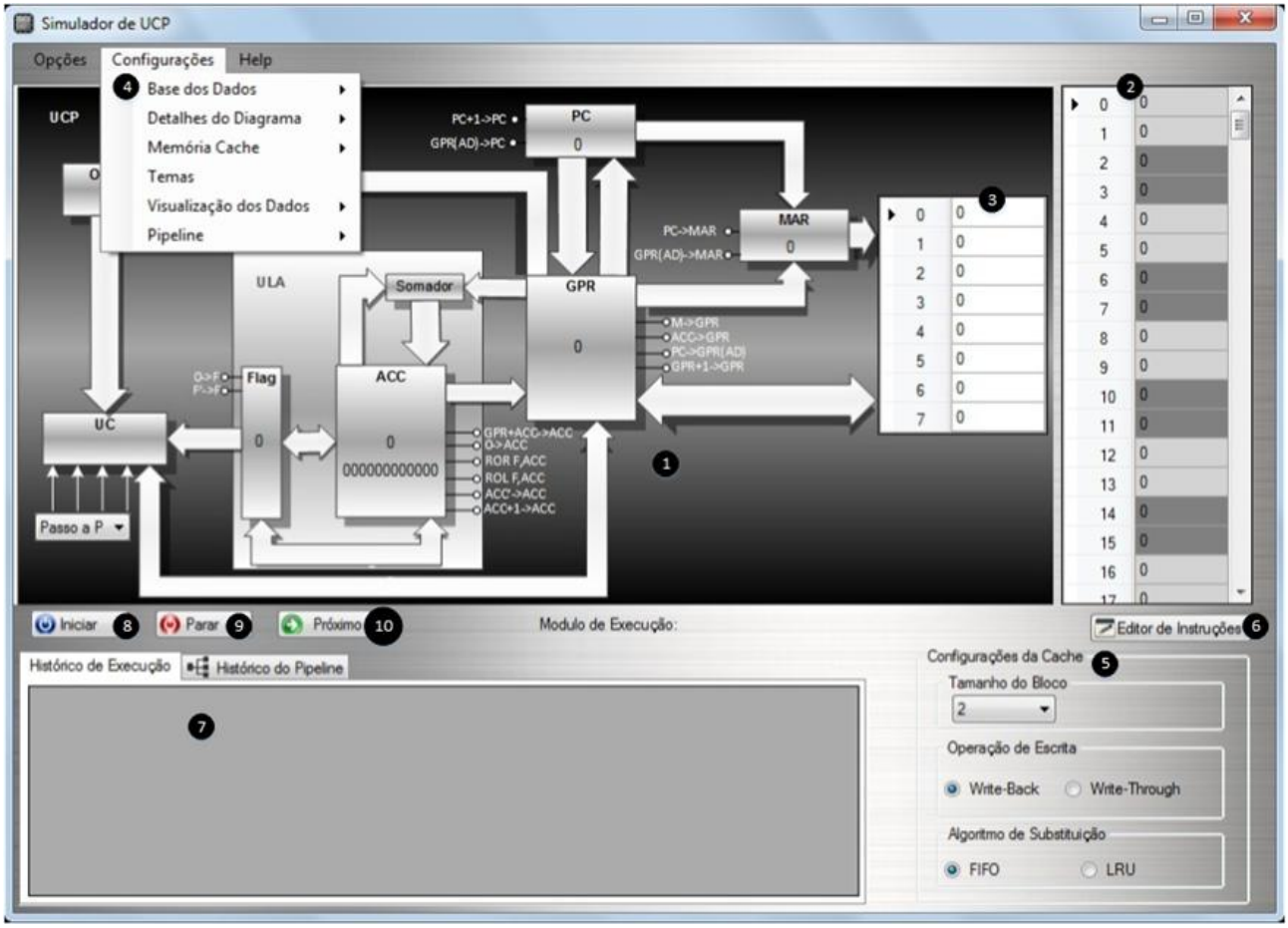
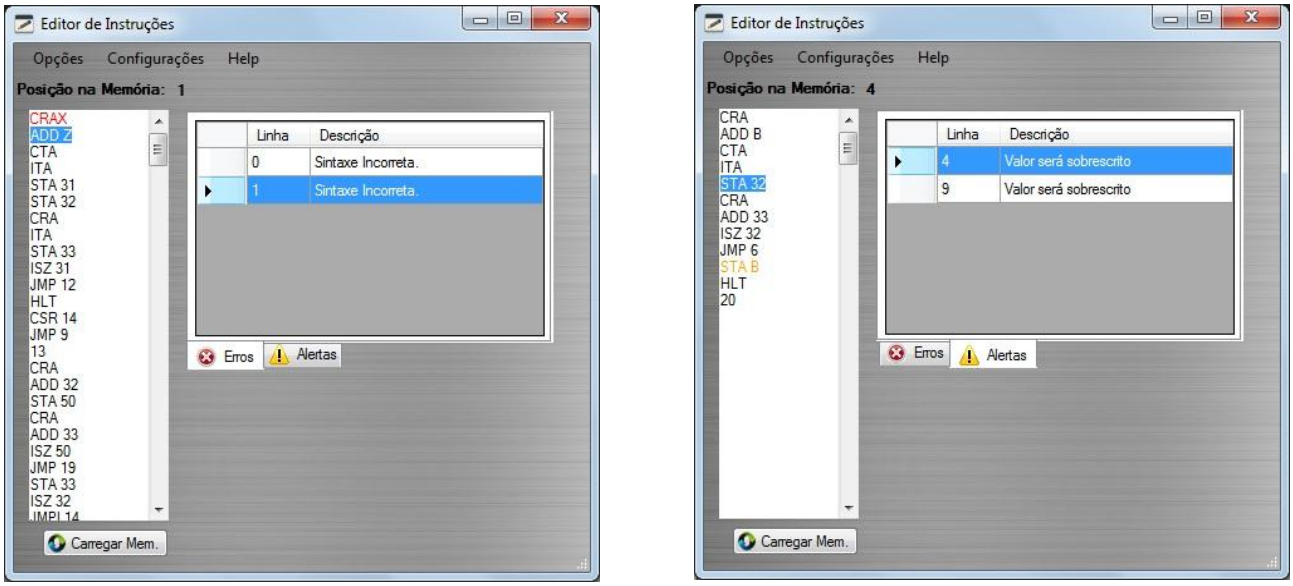


Figura 4. Simulador de UCP com suas funcionalidades numeradas.

As instruções são inseridas no formato de mnemônicos, diretamente na memória principal ou por meio de um editor de instruções. Nesse editor, a fim de auxiliar o desenvolvimento do programa, é realizada uma análise sintática e semântica simples, que indicam erros, como a escrita de comandos inválidos ou alertas como, por exemplo, quando se identifica instruções que irão sobrescrever outras instruções, ou a verificação de endereços que deveriam ter valores para que o processamento ocorra, mas que até o momento não foram inseridos pelo usuário. Por meio da Figura 5 é possível visualizar a interface do editor de instruções e alguns tratamentos sobre o código inserido. Caso o código possua instruções ou valores desconhecidos pelo simulador, sua execução não será possível até que sua sintaxe esteja correta. Além disso, permite que o usuário trabalhe com endereçamento e dados, tanto na base decimal quanto na hexadecimal.



(a) (b)

Figura 5. Editor de Instruções; (a) tratamento dos erros de sintaxe; (b) alertas sobre o código inserido.

A combinação de cores do diagrama de UCP pode ser alterada pelo usuário antes da execução do programa. Há alteração somente na aparência da UCP e na coloração dos barramentos, sendo preservada a arquitetura. Na Figura 6 são mostrados os temas presentes no simulador e na Figura 7, a interface com o tema selecionado.



Figura 6. Temas do simulador da UCP.

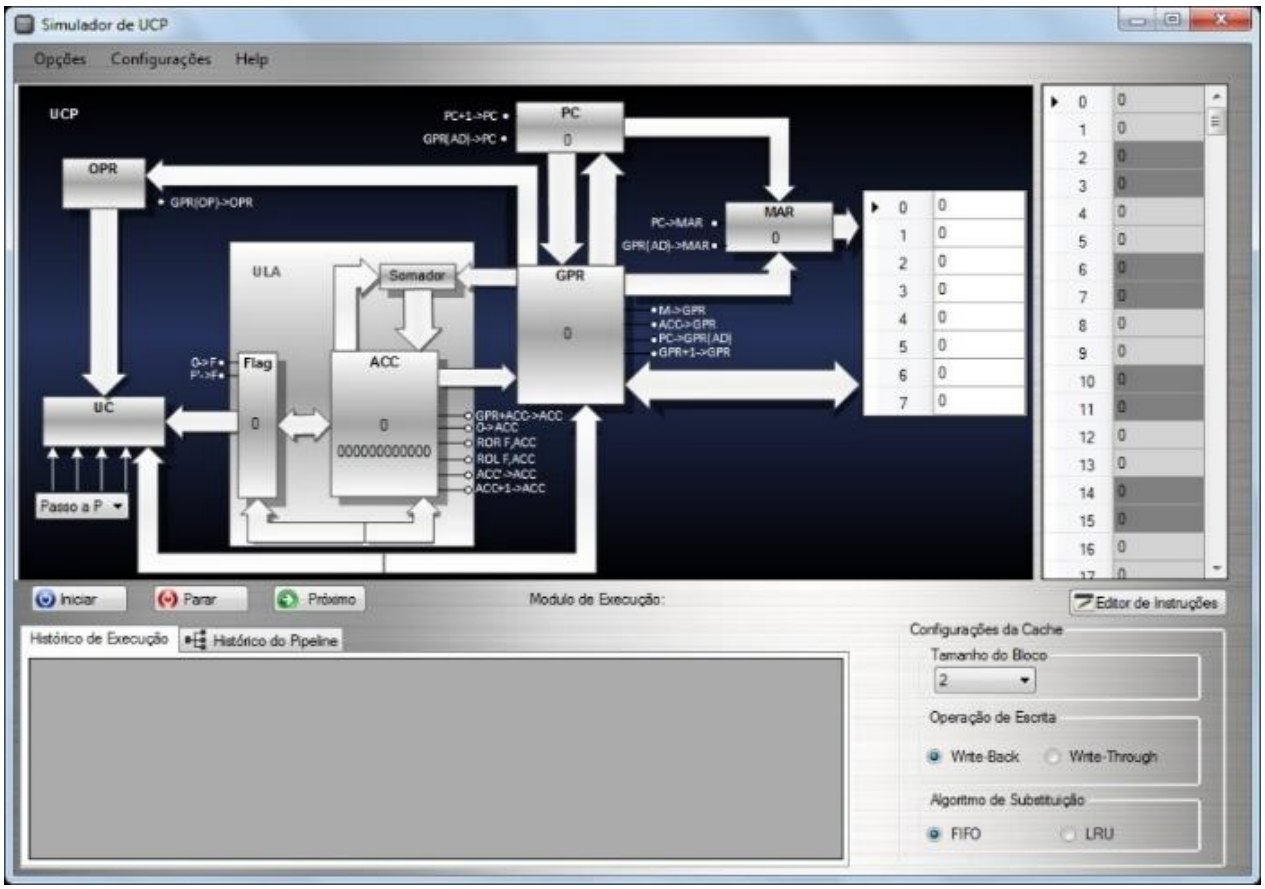


Figura 7. Design após escolha do novo tema do simulador da UCP.

Além disso, por se tratar de um simulador para fins didáticos, a ferramenta permite a visualização da função de cada componente que compõe a UCP conforme mostrado na Figura 8.

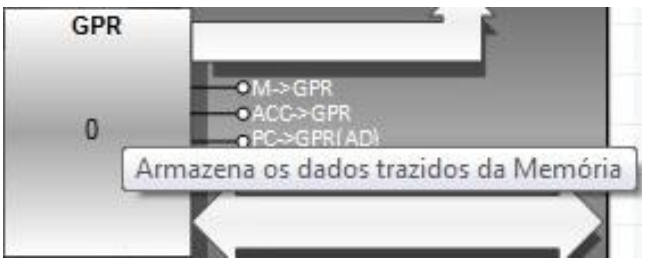
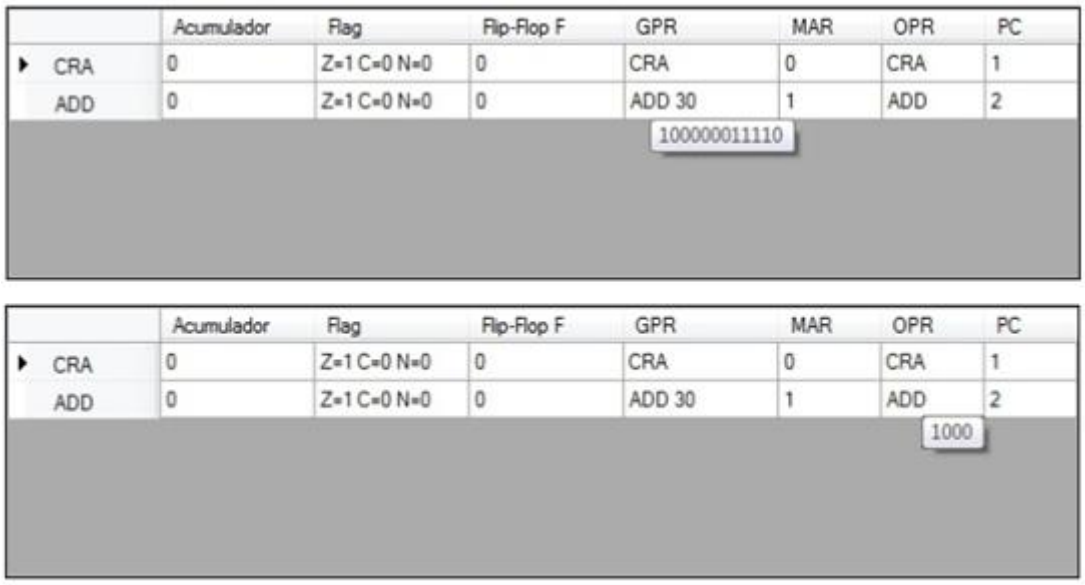


Figura 8. Informação sobre os componentes da UCP.

A análise da transferência dos bits do GPR para OPR é facilitada quando se configura a visualização dos dados na forma binária.

No histórico de instruções também é possível ver o valor dos registradores ACC, GPR e OPR na base binária como mostrado na Figura 9.



	Acumulador	Flag	Flip-Flop F	GPR	MAR	OPR	PC
▸ CRA	0	Z=1 C=0 N=0	0	CRA	0	CRA	1
ADD	0	Z=1 C=0 N=0	0	ADD 30	1	ADD	2

100000011110

	Acumulador	Flag	Flip-Flop F	GPR	MAR	OPR	PC
▸ CRA	0	Z=1 C=0 N=0	0	CRA	0	CRA	1
ADD	0	Z=1 C=0 N=0	0	ADD 30	1	ADD	2

1000

Figura 9. Visualização do conteúdo dos registradores na base binária.

O simulador possui suporte para memória *cache* de oito posições, permitindo escolher o tamanho da linha de *cache* tendo as opções: Linha de tamanho 2, Linha de tamanho 4 e Linha de tamanho 8. Uma vez selecionado o tamanho da linha de *cache*, a memória principal é agrupada nas cores branco e cinza, a fim de facilitar a visualização dos blocos de instrução a serem usados nas trocas (Figura 10).



Linha de tamanho 2

▸ 0	CRA
1	ADD 30
2	CTA
3	ITA
4	STA 31
5	STA 32
6	CRA
7	ITA
8	STA 33
9	ISZ 31
10	JMP 12
11	HLT
12	CSR 14
13	JMP 9
14	13
15	CRA
16	ADD 32
17	STA 50

Linha de tamanho 4

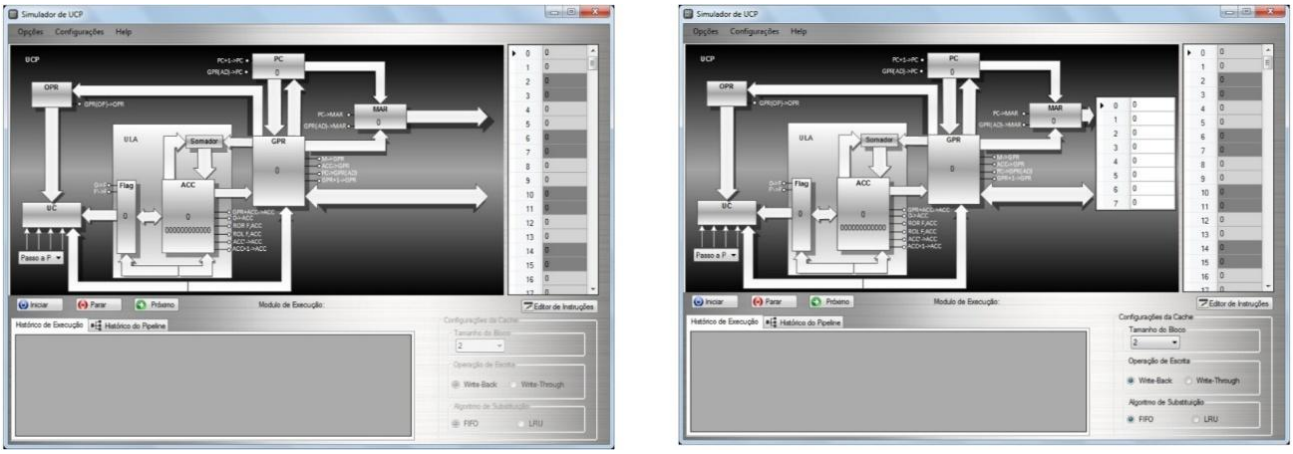
▸ 0	CRA
1	ADD 30
2	CTA
3	ITA
4	STA 31
5	STA 32
6	CRA
7	ITA
8	STA 33
9	ISZ 31
10	JMP 12
11	HLT
12	CSR 14
13	JMP 9
14	13
15	CRA
16	ADD 32
17	STA 50

Linha de tamanho 8

▸ 0	CRA
1	ADD 30
2	CTA
3	ITA
4	STA 31
5	STA 32
6	CRA
7	ITA
8	STA 33
9	ISZ 31
10	JMP 12
11	HLT
12	CSR 14
13	JMP 9
14	13
15	CRA
16	ADD 32
17	STA 50

Figura 10. Memória principal separada em linhas de *cache* (tamanho 2, tamanho 4 e tamanho 8).

Quando o simulador está com a configuração de memória *cache* desabilitada, o barramento de dados é ligado diretamente à memória principal com mostrado na Figura 11.



(a) (b)

Figura 11. Representação do uso de memória *cache*; (a) simulador sem a utilização de memória *cache*; (b) simulador com a utilização de memória *cache*.

Os algoritmos de substituição de bloco podem ser selecionados por FIFO (Figura 12(a)) ou LRU (Figura 12(b)). Na figura é apresentado o esquema de troca de informações da cache durante a execução de um mesmo programa com a utilização de cada um desses algoritmos. A análise de prioridade do bloco a ser substituído é feita por meio de cores, sendo que as mais claras representam os blocos com menor prioridade de substituição e as mais escuras, blocos com maior prioridade. Os marcadores representam, em ordem decrescente, a prioridade na escolha do bloco.

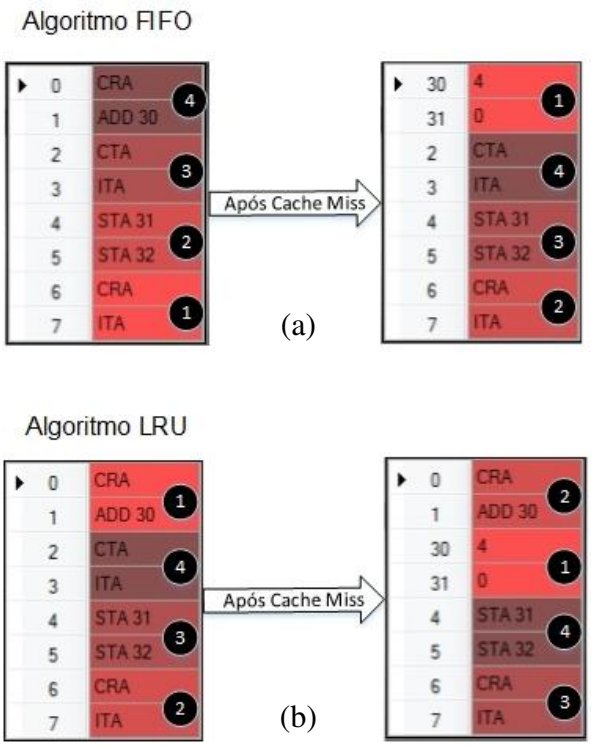
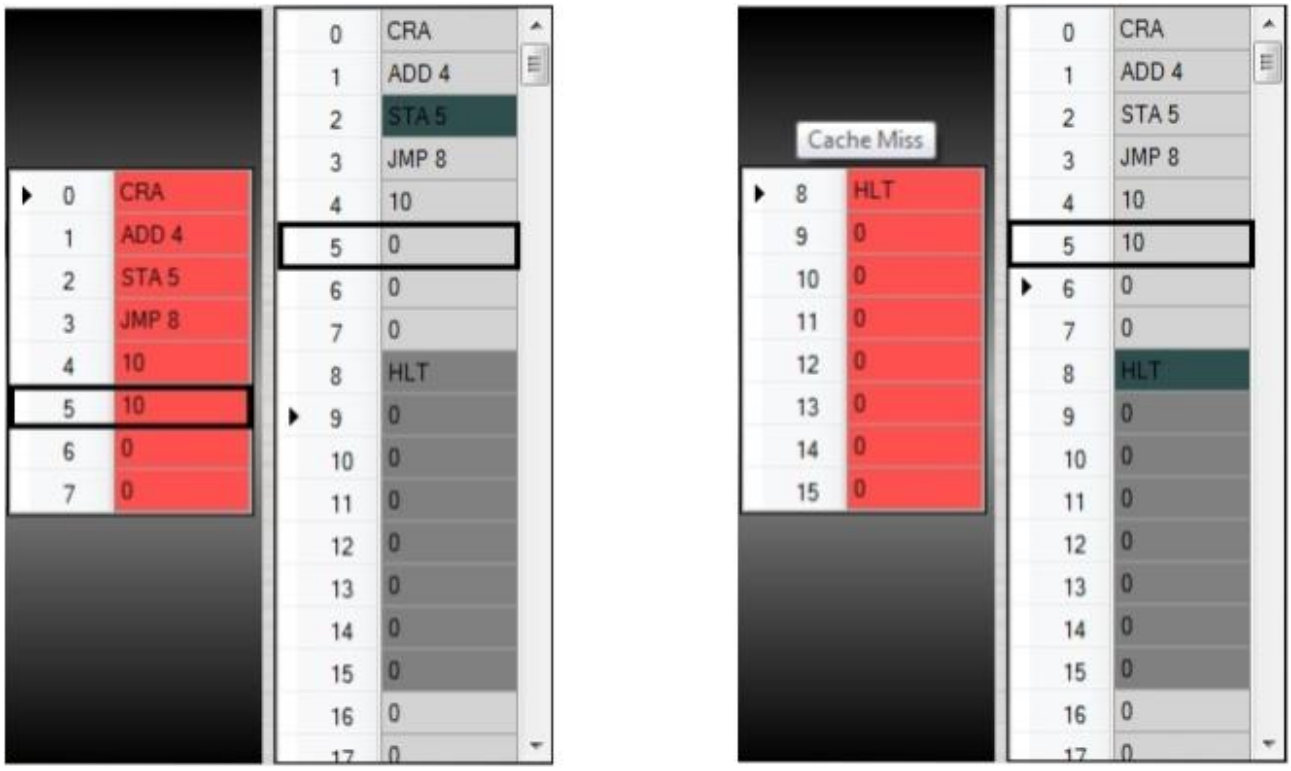


Figura 12. Simulador com a utilização dos algoritmos LRU e FIFO.

As técnicas de atualização da memória principal Write-Through e Write-Back também podem ser configuradas antes do início da execução. O funcionamento do método Write-Back é ilustrado na Figura 13. Após o *cache miss*, se valor alterado estiver no bloco que será substituído, a memória principal será atualizada quando o programa for finalizado, todos os valores que estão na *cache* também serão atualizados na memória principal. Para o método Write-Through, quando o dado da *cache* tem seu valor alterado imediatamente, a memória principal é atualizada.



(a) (b)

Figura 13. Técnica Write-Back; (a) memória principal com dado desatualizado; (b) memória principal com dado atualizado após substituição do bloco.

No momento em que as instruções vão sendo executadas é possível acompanhar todo o fluxo do processo por meio de animação gráfica.

O processo de execução de uma instrução quando o simulador não está utilizando *pipeline* é apresentado parcialmente, dividido em microinstruções e com coloração única para representar quais barramentos estão sendo utilizados. Para a *pipeline* foram utilizados dois estágios, busca e execução. No caso de sua utilização, o processo de cada estágio é mostrado de forma completa, para representar as instruções que estão sendo executadas em paralelo, uma cor diferente é atribuída para

cada estágio. Na Figura 14 é mostrada a diferença entre a execução de um mesmo programa com e sem a utilização da *pipeline*. Conforme o programa é executado, o estado dos registradores são gravados em um histórico de execução, possibilitando se ter conhecimento do seu estado quando executada uma determinada tarefa. As técnicas da *pipeline* (esvaziamento e ocioso) também são gravadas no histórico de *pipeline*, o que possibilita analisar a tomada de decisão da técnica de acordo com a instrução em execução. Para uma melhor visualização dos procedimentos realizados pela *pipeline*, o histórico pode ser maximizado e seu acompanhamento pode ser feito em tempo real (Figura 15).



(a) (b)

Figura 14. Recurso da *pipeline*; (a) simulador sem a utilização da *pipeline*; (b) simulador com a utilização da *pipeline*.



Figura 15. Histórico maximizado.

Todas as configurações possíveis para o simulador devem ser pré-estabelecidas antes do início da execução do código inserido, pois não é permitida a modificação do programa durante sua execução. O simulador foi desenvolvido na linguagem de programação C# utilizando a plataforma Visual Studio 2010. As imagens do diagrama de UCP foram elaboradas na ferramenta Microsoft Visio 2013.

5 RESULTADOS

O resultado obtido com a ferramenta foi considerado satisfatório, por permitir o acompanhamento detalhado de cada etapa que ocorre quando uma instrução é executada dentro da UCP.

A facilidade na análise de qual bloco da *cache* será escolhido para substituição e das técnicas de atualização da memória principal permitem uma melhor compreensão do recurso. A visualização, em tempo real, do que ocorre quando a *pipeline* está sendo utilizada facilita o entendimento das vantagens e desvantagens da técnica.

A variedade de configurações que o simulador permite o torna uma ferramenta capaz de servir como apoio em diferentes tópicos da área de arquitetura de computador.

Na Tabela 2 é mostrada uma comparação entre o simulador desenvolvido e os demais simuladores estudados nessa pesquisa. Levou-se em consideração a flexibilidade em alterar o repertório de instruções, o procedimento para realizar a inserção de instruções, o tratamento de erros de sintaxe e alertas, levantamentos estatísticos sobre a execução do programa como, por exemplo, o quanto de *clock* é utilizado para executar determinado programa e, por fim, recursos extras, como memória *cache* e *pipeline*.

6 CONSIDERAÇÕES FINAIS

O simulador é capaz de em um mesmo ambiente simular os recursos de memória *cache* e *pipeline*. A facilidade em usar um aplicativo para fins didáticos é um dos principais focos em simuladores, o que nesse projeto foi levado em consideração a fim de tornar o uso simples e eficiente. Com interface amigável de fácil manipulação, o simulador mostrou-se uma ferramenta capaz de auxiliar os docentes nas disciplinas de arquiteturas de computadores, bem como facilitar a compreensão dos alunos sobre os conceitos apresentados.

Como trabalhos futuros, pode-se citar: a) mais estágios da *pipeline* e aplicação de mais técnicas que evitem seu esvaziamento como, por exemplo, a utilização de execução especulativa e concorrente; b) a realização de uma análise estatística que compare em

termos de quantidade de *clock* usada para execução com e sem os recursos da *pipeline* e memória; c) a criação de processos de interrupção de dispositivos de entrada e saída; d) criação de instruções que agreguem um conjunto de microinstruções possibilitando o uso em um nível mais alto do Assembly.

Tabela 2. Comparativo entre os simuladores.

	Simulador de UCP	Sistema Simulador de Arquitetura Básica de Processadores	Simulador de uma Arquitetura Básica de Computadores	UCP-8E
Alteração das instruções	Não	Não	Sim	Não
Editor de instruções	Sim	Sim	Não	Não
Análise de erros	Sim	Não	Não	Não
Pipeline	Sim	Não	Não	Não
Cache	Sim	Não	Não	Não
Análise Estatística da execução	Não	Não	Não	Não

REFERÊNCIAS

SANTOS, S. S. F. **Sistema simulador de arquitetura básica de processadores**. 2005. Monografia (Ciência da Computação) – Faculdade de Informática de Presidente Prudente, Universidade do Oeste Paulista, Presidente Prudente, SP.

SILVA-FILHO, J. G. **UCP-8E-v2.2 uma UCP educacional de 8-bits**. Disponível em: <https://code.google.com/p/cpu-8e/>. Acesso em: abr. 2013.

SITOLINO, C. L. **Sistema simulador de uma arquitetura básica**. 1992. Monografia (Ciência da Computação) – Faculdade de Informática de Presidente Prudente, Universidade do Oeste Paulista, Presidente Prudente, SP.

STALLINGS, W. **Arquitetura e organização de computadores**. 5. ed. São Paulo: Pearson Prentice-Hall, 2003.

TANENBAUM, A. S. **Organização estruturada de computadores.** 5. ed. São Paulo: Pearson Prentice-Hall, 2007.

TAUB, H. **Circuitos digitais e microprocessadores**. São Paulo: McGraw-Hill do Brasil, 1984.