# 2º Trabalho do semestre

# Construção de um emulador do MIPS 3000

Descrição: Neste trabalho o aluno deve implementar utilizando a linguagem de programação de sua preferência um programa que emule o funcionamento do processador MIPS 3000.

**Requisitos:**

① O emulador deve ser capaz de executar as seguintes instruções:

| | | |
|---|---|---|
| 01 | add | add rd,rs,rt |
| 02 | sub | sub rd,rs,rt |
| 03 | div | div rd,rs,rt |
| 04 | mul | mul rd,rs,rt |
| 05 | addi | addi rd,rs,cte |
| 06 | and | and rd,rs,rt |
| 07 | or | or rd,rs,rt |
| 08 | nor | nor rd,rs,rt |
| 09 | xor | xor rd,rs,rt |
| 10 | andi | andi rt,rs,cte |
| 11 | ori | ori rt,rs,cte |
| 12 | sll | sll rt,rs,cte |
| 13 | slr | slr rt,rs,cte |
| 14 | beq | beq rt,rs,LABEL |
| 15 | bne | bne rt,rs,LABEL |
| 16 | slt | slt rd,rs,rt |
| 17 | j | j LABEL |
| 18 | jr | jr rs |
| 19 | jal | jal LABEL |
| 20 | lw | lw rt,cte(rs) |
| 21 | sw | sw rt,cte(rs) |
| 22 | syscall | syscall |
| 23 | mfh | mfh rs |
| 24 | mfl | mfl rs |

② O emulador deve ser capaz de emular uma memória RAM de 64k words de 32 bits. Ela pode ser implementada por exemplo em linguagem C da seguinte forma:

```
long RAM[65536];
```

③ O Emulador deve possuir uma função para carregamento de programas. Os programas possuem a forma de um arquivo binário contendo o código de máquina das instruções. O código de máquinas será carregado na RAM a partir do endereço $0000_H$. Caso o programa possua um segmento de dados, este será fornecido em um arquivo separado. Os dados do programa devem ser carregados a partir do primeiro word disponível após o carregamento do programa.

④ O esquema de nomeação dos arquivos de código de máquina e dados de um programa segue a seguinte configuração:

```
<programa>.asm ← código de máquina

<programa>.dat ← dados do programa
```

⑤O código de máquina das instruções pode ser levantado a partir do emulador MARS. Escreva por exemplo um programa sem sentido que possua todas as instruções listadas na tabela acima. Compile o programa e utilize o modo de execução passo a passo para levantar o código de máquina de cada instrução. **Produza um documento contendo uma tabela que lista o código de máquina de cada instrução**.

⑥ O esquema de numeração dos registradores seguirá o mesmo esquema adotado no emulador MARS.

⑦ O programa deve possuir uma função para listar seguimentos de memória. Por exemplo:

```
>> list 0000H – 0040H

0000H 0000 0000 0000 0000 0000 0000 0000 0000
0008H 0000 0000 0000 0000 0000 0000 0000 0000
0010H 0000 0000 0000 0000 0000 0000 0000 0000
0018H 0000 0000 0000 0000 0000 0000 0000 0000
0020H 0000 0000 0000 0000 0000 0000 0000 0000
0028H 0000 0000 0000 0000 0000 0000 0000 0000
0030H 0000 0000 0000 0000 0000 0000 0000 0000
0038H 0000 0000 0000 0000 0000 0000 0000 0000
0040H 0000
```

⑧ O programa deve possuir uma função que execute o programa a partir da primeira instrução até encontrar a instrução `syscall` com o código de interrupção `10H`.

⑨ O programa deve implementar as interrupções previstas na tabela abaixo:

| Serviço | Cod. $v0 | Argumentos | Resultado |
|---|---|---|---|
| imprimir inteiro | 1 | $a0 inteiro a ser impresso | |
| imprimir string | 4 | $a0 endereço para a string terminada em NULL | |
| ler inteiro | 5 | | $v0 contém o inteiro lido |
| ler string | 8 | | $a0 endereço do início do buffer. $a1 num max de bytes lidos |
| terminar programa | 10 | | |

⑩ O Emulador deve possuir uma função para execução passo a passo do programa carregado em memória. Após a execução de uma instrução, o emulador deve permitir que o conteúdo da memória seja inspecionado caso se deseje;

⑪ O Emulador deve permitir que ele seja terminado a qualquer momento;

⑫ A execução do emulador sem a carga prévia de um programa deve ser identificada e apropriadamente informada;

⑬ O Emulador deve permitir a inspeção do conteúdo dos registradores a qualquer momento;

```
>> reg 1
>> 440000FF_H
```

⑭ fornecer uma opção para consulta de todos os registradores de uma vez;

```
>> regs
00 00000000 00000000 00000000 00000000
04 00000000 00000000 00000000 00000000
08 00000000 00000000 00000000 00000000
12 00000000 00000000 00000000 00000000
16 00000000 00000000 00000000 00000000
20 00000000 00000000 00000000 00000000
24 00000000 00000000 00000000 00000000
28 00000000 00000000 00000000 00000000
```

⑮ todas as saídas de dados (memória, registradores, etc) devem ser feitas impreterivelmente em hexadecimal;

**Condições**:

- Apresentação impreterível em 20/06/2016;
- A não execução correta dos três programas de teste fornecidos a priori implica num corte em 50% da nota total;
- A execução correta implica em 50% da nota creditada, provido que nenhuma instância de plágio tenha sido identificada;
- Os 50% restantes da nota do trabalho serão creditados aos outros requisitos previstos nesta descrição de trabalho e mediante a avaliação do código fonte do emulador a discrição do professor;
- Trabalho deve ser executado em duplas; A formação de duplas ficam a cargo dos alunos; não serão aceitos trabalhos individuais nem em trios, quartetos, etc;
- A formação de duplas deve ser informada ao professor até o dia 06/06/2016. A eventual possibilidade de um aluno ficar sozinho será estudada e tratada de acordo no dia da definição final de duplas;
- Os horários de atendimento podem ser utilizados para sanar dúvidas dos trabalhos;
- Cópias de trabalhos serão anuladas. Caso o professor identifique cópia parcial ou total de trabalhos os alunos terão uma chance de se explicarem e provarem

o entendimento do conteúdo do trabalho via uma prova horal, na qual o professor postulará diversas perguntas individuais aos alunos em disputa. Se ficar provada a inadequação da assimilação do conteúdo do trabalho, a nota será zerada. Recursos quanto a anulação do trabalho deverão impreterivelmente seguir os ritos previstos nas normas de graduação da universidade.

**Referências**:


① MARS. MIPS Assembler and Runtime Simulator. Missouri State University. Disponível em: http://courses.missouristate.edu/KenVollmar/MARS/

② CCSC-MW paper, "A MIPS Assembly Language Simulator Designed for Education." Ken Vollmar and Pete Sanderson. Journal of Computing Sciences in Colleges, 21:1, October 2005. Pages: 95 - 101.

③ SIGCSE 2006 paper, "MARS: An Education-Oriented MIPS Assembly Language Simulator," Kenneth Vollmar and Pete Sanderson. ACM SIGCSE Bulletin, 38:1 (March 2006), 239-243.

④ MARS presentations by Pete Sanderson at Bowling Green State Univ. (2006) and University of Pittsburgh (2007).

⑤ Tutorial on MARS at CCSC-CP, Drury University, Apr. 13-14, 2007, by Pete Sanderson and Ken Vollmar.

⑥ PATTERSON, D. A. e HENNESSY, J. L. 2014. Organização e Projeto de Computadores – A Interface Hardware/Software. Elsevier/ Campus 4ª edição.

⑦ HENNESSY, J. L. e PATTERSON, D. A. 2012. Arquitetura de Computadores – Uma Abordagem Quantitativa. Elsevier/ Campus 5ª edição.

⑧ IDT R30xx Family. Software Reference Manual Disponível em: https://cgi.cse.unsw.edu.au/~cs3231/doc/R3000.pdf

⑨ MIPS32™ Architecture For Programmers Volume I: Introduction to the MIPS32™ Architecture. Disponível em: http://www.cs.cornell.edu/courses/cs3410/2008fa/mips_vol1.pdf

⑩ MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set. Disponível em: http://www.cs.cornell.edu/courses/cs3410/2008fa/mips_vol2.pdf