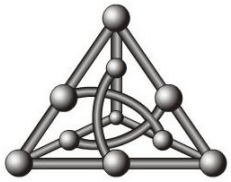

Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Disciplina: Arquitetura de Computadores II
Professor: Renan Albuquerque Marks

Descrição do Trabalho Prático

1 Introdução

Neste documento estão detalhados os procedimentos que devem ser seguidos para o desenvolvimento do Trabalho da disciplina de Arquitetura de Computadores II. Este trabalho será desenvolvido ao longo deste semestre letivo e constituirá como parte da nota final da disciplina de Arquitetura de Computadores II.

É fortemente recomendado que os estudantes acessem com freqüência este documento para esclarecer possíveis dúvidas, estar ciente do cronograma e estar a par de possíveis atualizações/alterações no trabalho.

2 Objetivo

Desenvolver o principal componente de um emulador de Playstation 2®: um emulador funcional de um processador MIPS. O seu emulador deve ser capaz de executar um binário da ISA MIPS. Este emulador possuirá a capacidade de emular um conjunto reduzido de instruções MIPS, pois quer-se verificar a viabilidade de um novo emulador da plataforma PS2 que seja mais rápido e consuma menos recursos, capaz de rodar em qualquer dispositivo móvel/embarcado (principalmente em smartcars Tesla® e smartwatches Apple® e Samsung®).

3 O que deve ser feito?

Desenvolver um emulador funcional do processador MIPS em linguagem **Python 3**, **C/C++** ou **Java** que implemente o comportamento das instruções listadas na Tabela 1. O programa **deve** vir com um script de compilação (makefile ou script bash comum) que realize a compilação e geração de executável do emulador. O emulador deve conseguir ser compilado pelo compilador GCC/G++ (caso C/C++), JavaC (caso Java) ou executado pelo interpretador Python 3 padrão (CPython) **sem nenhum erro ou *warning***.

Um emulador funcional do processador MIPS deve ser capaz de simular o comportamento das instruções executadas pelo processador sem haver necessidade de simular detalhes da microarquitetura (como quantidade de ciclos por instrução, pipelines, sinais de controle, etc). Dessa forma, emuladores funcionais devem simular:

- Armazenamento em registradores;
- Armazenamento em memória:
  - Endereçamento por bytes;
  - Seção de texto;
  - Seção de dados globais;
  - Pilha;
- Registradores ocultos (PC, HILO);

- Comportamento de instruções (descritas na Tabela 1);
- Chamada e retorno de funções (instruções `jal` e `jr`);

Categoria	Nome	Sintaxe	Formato	Comportamento
Aritmética	Add	`add $d,$s,$t`	R	$d = s + t$
	Substract	`sub $d,$s,$t`	R	$d = s - t$
	Add Immediate	`addi $d,$s,C`	I	$d = s + C$
	Add Immediate Unsigned	`addiu $d,$s,C`	I	$d = s + C$(não considera compl_2)
	Multiply	`mult $s,$t`	R	$HILO = s \times t$
	Divide	`div $s,$t`	R	$LO = s \div t$; $HI = s \mod t$;
Transferência de dados	Load Word	`lw $t,C($s)`	I	$t =$ Memória$[C + s]$ (4 bytes)
	Load Halfword	`lh $t,C($s)`	I	$t =$ Memória$[C + s]$ (2 bytes)
	Load Byte	`lb $t,C($s)`	I	$t =$ Memória$[C + s]$ (1 byte)
	Store Word	`sw $t,C($s)`	I	Memória$[C + s] = t$ (4 bytes)
	Store Halfword	`sh $t,C($s)`	I	Memória$[C + s] = t$ (2 bytes)
	Store Byte	`sb $t,C($s)`	I	Memória$[C + s] = t$ (1 bytes)
	Load Upper Immediate	`lui $d,C`	I	$d = C \ll 16$
	Move from high	`mfhi $d`	R	$d = HI$
	Move from low	`mflo $d`	R	$d = LO$
Lógica	And	`and $d,$s,$t`	R	$d = s\&t$
	And Immediate	`andi $d,$s,C`	I	$d = s\&C$
	Or	`or $d,$s,$t`	R	$d = s\|t$
	Or Immediate	`ori $d,$s,C`	I	$d = s\|C$
	Set on Less Than	`slt $d,$s,$t`	R	$d = (s < t)?1 : 0$
	Set on Less Than Immediate	`slti $d,$s,C`	I	$d = (s < C)?1 : 0$
Deslocamento de bits	Shift left logical	`sll $d,$t,shamt`	R	$d = t \ll shamt$
	Shift right logical	`srl $d,$t,shamt`	R	$d = t \gg shamt$
	Shift right arithmetic	`sra $d,$t,shamt`	R	$d = t \gg shamt$ (com ext. sinal)
Desvio condicional	Branch on equal	`beq $s,$t,C`	I	if (s == t) $PC = $PC + 4 + (C*4)
	Branch on not equal	`bne $s,$t,C`	I	if (s != t) $PC = $PC + 4 + (C*4)
Desvio incondicional	Jump	`j C`	J	$\$PC = (\$PC[31 - 28] + 4) \| (C \ll 2)$
	Jump register	`jr $s`	R	$\$PC = s$
	Jump and link	`jal C`	J	$\$RA = \$PC + 4; \$PC = (\$PC + 4)[31 - 28] \| (C \ll 2)$

Tabela 1: Instruções MIPS a serem implementadas e seus respectivos comportamentos.

4 Entrada

A entrada terá as seguintes características:

- As entradas do emulador serão:
  - Um **arquivo binário** contendo a seção de texto (até 4 KibiBytes de texto);
  - Um **arquivo binário** contendo a seção de dados globais(até 4 KibiBytes de dados globais);
- O emulador será executado com a seguinte linha de comando:
  - Caso seja um script Python 3: `./python3 text.bin data.bin > saida.txt`
  - Caso seja um binário C/C++: `./emulador text.bin data.bin > saida.txt`
  - Caso seja um binário Java: `java -jar emulador.jar text.bin data.bin > saida.txt`

5 Saída

A saída do emulador **deve ser** de texto no console/terminal (saída-padrão) e **deve conter** o valor de todos os 32 registradores visíveis (em hexadecimal) e todo o conteúdo da memória (em hexadecimal) no mesmo formato da Figura 3.

Como gabarito para checar a corretude de seu emulador, a mesma saída pode ser obtida ao executar o simulador MARS especificando as **opções via linha de comando** que podem ser vistas na Figura 1.

```
java -jar Mars4_5.jar nc mc CompactTextAtZero $0 $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12 $13
$14 $15 $16 $17 $18 $19 $20 $21 $22 $23 $24 $25 $26 $27 $28 $29 $30 $31 0x0-0x3ffc entrada.asm
```

Figura 1: Exemplo de execução em linha de comando do MARS para obtenção da saída presente na Figura 3.

6 Detalhes técnicos do emulador

Os arquivos-teste serão produzidos a partir do comando "Dump Memory" presente no menu "File" do simulador MARS com o mapa de memória "Compact, text at address 0". Este mapa de memória deve ser selecionado no menu "Settings" e "Memory Configuration".

Dessa forma, o emulador deverá seguir o mesmo mapa de memória presente na configuração "Compact, text at address 0" do simulador MARS. Nesse mapa de memória, o emulador deverá ter uma memória principal de 16 KibiBytes de capacidade de armazenamento que deve deve ser inicializada com zero em todos os bytes.

Ou seja, as seções de texto e dados deverão ter as seguintes características:

- Seção de texto (endereços `0x0000` até `0x0fff`):
  - Tamanho da seção será fixo de 4096 bytes (4 KibiBytes);
  - Memória de toda a seção deverá ser inicializada com zero (mesmo que o programa carregado não ocupe todo o espaço reservado para a seção);
  - A seção será alocada na memória a partir do endereço `0x0` até `0x0fff`;
  - Programa organizado em funções;
  - O programa sempre deverá começar carregado a partir do endereço `0x0`;
- Seção de dados (endereços `0x2000` até `0x2fff`):
  - Tamanho da seção será fixo de 4096 bytes (4 KibiBytes);
  - Registrador `$gp` deve ser inicializado com `0x1800`;
  - Memória de toda a seção deverá ser inicializada com zero (mesmo que o programa carregado não ocupe todo o espaço reservado para a seção);
  - A seção terá os dados alocados na memória a partir do endereço `0x2000` até `0x2fff`;
  - Contém somente dados (strings, valores imediatos, constantes, etc);
- Pilha (endereços `0x3000` até `0x3fff`):
  - Tamanho da seção será fixo de 4096 bytes (4 KibiBytes);
  - Registrador `$sp` deve ser inicializado com `0x00003ffc`;
  - A pilha poderá trabalhar na região de memória entre os endereços `0x3000` até `0x3fff`;
  - Caso o topo da pilha ultrapasse o endereço limitante `0x3000`, o emulador deve encerrar a emulação com a mensagem "Error: stack overflow.";

Uma forma de emulação da memória física em software é utilizando-se um vetor de bytes na linguagem de programação escolhida para o desenvolvimento do trabalho. Um exemplo de implementação nas linguagens C/C++ e Java é através de `char memory[4096 * 4];`.

O emulador também deve suportar algumas chamadas de sistema. As chamadas de sistema a serem implementadas são as mesmas presentes no simulador MARS. São elas:

- Syscalls 1, 4, 5 e 8;
- Syscalls de 10 a 12;
- Syscalls de 34 a 36;

Seu emulador deve ser 100% compatível com as mesmas chamadas de sistema do simulador MARS. Para maiores informações a respeito das chamadas de sistema, veja http://courses.missouristate.edu/kenvollmar/mars/Help/SyscallHelp.html.

7 Testando e corrigindo seu próprio trabalho

O trabalho prático será corrigido usando o método descrito abaixo. O processo de correção será automatizado via scripts, sem interferência humana. Nos passos a seguir, um exemplo de como checar (e corrigir) a saída de seu programa a partir de um arquivo de entrada "`bubble.s`" que implementa o algoritmo de Bubble-Sort em assembly MIPS:

Passo 1 Utilize o MARS para gerar os arquivos binários de texto (instruções), dados e salvar em um arquivo texto toda a saída gerada pelo MARS, inclusive a gerada pelo seu próprio programa de teste quando executado (`bubble_out_mars.txt`):

```
java -jar ../Mars4_5.jar a dump .text Binary bubble_text.bin bubble.s

java -jar ../Mars4_5.jar dump .data Binary bubble_data.bin nc mc CompactTextAtZero
$0 $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12 $13 $14
$15 $16 $17 $18 $19 $20 $21 $22 $23 $24 $25 $26 $27
$28 $29 $30 $31 0x0-0x3ffc bubble.s > bubble_out_mars.txt
```

Passo 2 Execute o seu programa com os binários gerados pelo MARS e salve saída em outro arquivo texto (`bubble_out.txt`).

```
./emulador bubble_text.bin bubble_data.bin > bubble_out.txt
```

Passo 3 Compare ambas as saídas usando o software KDiff3. Não deve haver qualquer diferença entre o conteúdo de ambos os arquivos. Um exemplo de execução do KDiff3 pode ser visto na Figura 2.

```
kdiff3 bubble_out_mars.txt bubble_out.txt
```

8 Software demonstração

A submissão de um *software demo* também fará parte da nota final do trabalho. O código fonte do *demo* deve ser submetido e estar, no mínimo, em linguagem de montagem MIPS. Se o software demo for desenvolvido em outra linguagem (por exemplo, C), devem ser submetidas para avaliação tanto a versão em C quanto em assembly MIPS.

O *demo* deverá **ser capaz de rodar sem falhas em seu emulador MIPS**. Além disso, ele deve explorar os recursos implementados em seu emulador da melhor forma possível, ou seja: destacar os pontos fortes de forma a "vender" o seu "produto" para um determinado público alvo.

Como exemplos de softwares *demo* podem ser citados: desde algoritmos técnicos específicos (ordenação, cálculos matemáticos, etc) até aplicativos interativos (aplicativos para usuários comuns como editores de texto, gráficos, jogos simples ou complexos, etc).

Quanto melhor e/ou mais criativo o *demo* disponibilizado, melhor será a avaliação do trabalho desenvolvido.

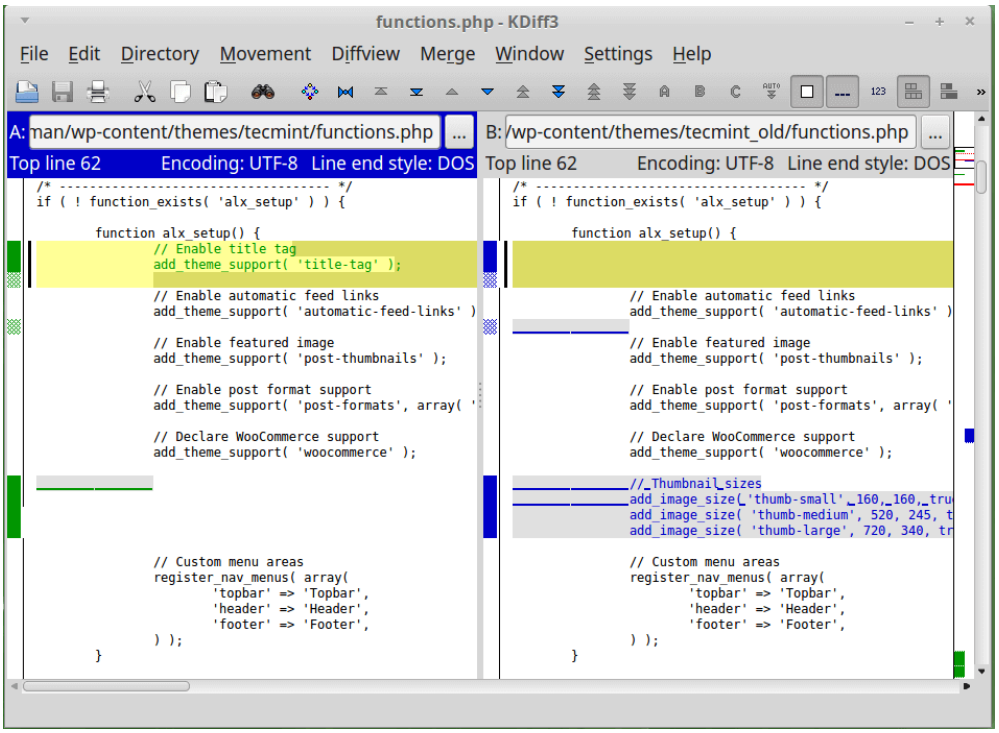

Figura 2: KDiff3 mostrando diferenças entre dois arquivos.

9 Atribuição de grupos

Os trabalhos deverão ser feitos por grupos contendo no máximo 2 (dois) participantes.

10 Cronograma

- **Início dos trabalhos:** 08/05/2023;
- **Entrega:** 09/06/2023 até as 23h59 — Cada grupo deve submeter, via Moodle, a versão final do código do emulador junto de um relatório que contenha:
  - Integrantes do grupo;
  - Explicação detalhada do funcionamento do emulador;
  - Funcionalidades implementadas:
    - Instruções;
    - Recursos;
    - Syscalls;
    - Soluções de implementação;
    - Organização do código-fonte (se houver mais de um arquivo);
  - Funcionalidades não implementadas:
    - Dificuldades encontradas;
  - **IMPORTANTE!** Inclua os casos de teste (código-fonte dos programas de teste) que você usou para testar seu emulador; Eles serão checados durante o processo de correção.

11 Avaliação do trabalho

- **Nota do Emulador:** valor no intervalo [0,5] que será atribuído **comparando-se as saídas-padrão do emulador e do MARS salvas em modo texto** após a execução de casos

de teste.

- **Nota do *Demo*:** valor no intervalo [0,5] que será atribuído após a execução de casos de teste.
- A nota final do trabalho prático será composta da soma da nota do emulador com a nota do demo submetidos por cada grupo;
- Caso o seu programa contenha mais de um arquivo .py/.c/.cpp/.java, **inclua também** um script de compilação ou MakeFile;
- Trabalhos que **não** compilarem e/ou executarem receberão nota **zero**;
- **Não será admitido** uso de bibliotecas de simulação/emulação externas no desenvolvimento do trabalho;
- **Atenção:** Não serão aceitas entregas de trabalho atrasadas.
- **Casos de Plágio nos códigos serão tratados com rigor**.

12 Dicas e Sugestões

- Inicie o trabalho o **quanto antes**. O tempo **voa**!
- Retire as dúvidas quanto ao entendimento dos elementos que compõem a arquitetura. Isso possibilitará detectar possíveis falhas na implementação logo cedo.
- Resultado da sua simulação poderá ser comparado com o resultado do simulador MARS (http://courses.missouristate.edu/kenvollmar/mars/);
- Trabalhem em equipe e implementem tanto o emulador quanto o *demo* em paralelo: uma equipe no emulador, outra no demo; frequentemente testem ambas as implementações juntas, equipes no mercado de trabalho que desenvolvem tanto hardware quanto software (Apple, Tesla, Intel, Google, Samsung, etc) utilizam uma metodologia de trabalho similar;
- Organize seu código de modo a que ele tenha partes simples e reusáveis: componha estruturas/classes complexas a partir de estruturas/classes mais simples;
- Caso sinta-se confortável, utilize a orientação a objetos ao seu favor!
- Se você já cursou (ou cursa) as disciplinas de Engenharia de Software aproveite para exercitar os conceitos de engenharia de software que facilitem seu trabalho: modelagem de software, arquitetura de software, métodos ágeis, programação em pares/trios, controle de versão (Git, SVN, etc);

```
$0      0x00000000
$1      0x00000000
$2      0x0000000a
$3      0x00000000
$4      0x00002004
$5      0x00000000
$6      0x00000000
$7      0x00000000
$8      0x00000050
$9      0x00002050

...

$26     0x00000000
$27     0x00000000
$28     0x00001800
$29     0x00003ffc
$30     0x00000000
$31     0x00000000
Mem[0x00000000] 0x23bdfffc      0xafbf0000      0x0c000007      0x8fbf0000
Mem[0x00000010] 0x23bd0004      0x2002000a      0x0000000c      0x23bdfffc
Mem[0x00000020] 0xafbf0000      0x20042004      0x8c052000      0x0c000012
Mem[0x00000030] 0x20042004      0x8c052000      0x0c000049      0x8fbf0000
Mem[0x00000040] 0x23bd0004      0x03e00008      0x23bdfffc      0xafbf0000
Mem[0x00000050] 0x00004820      0x200a0001      0x0145602a      0x11800017
Mem[0x00000060] 0x01406820      0x000d6880      0x01a46820      0x8dae0000
Mem[0x00000070] 0x8daffffc      0x01cf602a      0x1180000e      0x23bdfff0

...

Mem[0x00003fc0] 0x00000000      0x00000000      0x00000000      0x00000000
Mem[0x00003fd0] 0x00000000      0x0000009c      0x00000000      0x00000000
Mem[0x00003fe0] 0x00002004      0x00000014      0x00000001      0x00002004
Mem[0x00003ff0] 0x0000003c      0x0000000c      0x00000000      0x00000000
```

Figura 3: Exemplo de saída do emulador. Para o espaçamento entre valores na mesma linha, utilize 1 caracter de tabulação (`\t`).