Trabalho Prático III: Simulador de Instruções do MIPS*

Prof. Pedro Henrique Penna

Graduação em Engenharia de Software – 2º Período
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

Resumo

Este trabalho faz parte de uma série que tem como objetivo final a construção de um simulador de uma máquina MIPS. Nesta última etapa etapa, você deverá se basear nos artefatos construídos anteriormente para efetivamente implementar o simulador.

---

1. Descrição

Você deverá escrever um programa que simula o funcionamento de uma máquina MIPS, dotada de Unidade Lógica e Aritmética, Banco de Registradores e Memória Principal. Seu simulador deverá ler comandos em linguagem de montagem de um arquivo de entrada e proceder da seguinte forma:

1. Decodificar o comando em linguagem de montagem para instrução de máquina do MIPS.
2. Imprimir a instrução correspondente no arquivo de saída `instructions.out`.
3. Atualizar o valor dos registradores envolvidos (se houver).
4. Atualizar o conteúdo da memória principal (se for o caso).

Após ler e executar o último comando do arquivo de entrada, seu simulador deve, por fim:

1. Escrever o conteúdo dos registradores no arquivo de saída `registers.out`.
2. Escrever o conteúdo da memória no arquivo de saída `memory.out`.

Para informações sobre o formato de cada um dos arquivos de entrada e saída do simulador, consulte os anexos do presente documento. Ademais, a máquina simulada deve:

- Ter uma memória de 16 kB (4096 linhas de 4 bytes cada).
- Possuir 32 registradores, nome e código especificados na Tabela 1.
- Suportar a execução das instruções do MIPS listadas na Tabela 2.

* Qualquer inconsistência de informação no enunciado deste trabalho com o livro-texto base da disciplina é não intencional. Em caso de dúvida, sempre recorra ao livro e converse com o professor a respeito. O presente enunciado está sujeito a correções nesse sentido e, caso ocorram, serão divulgadas no SGA e em sala.

Tabela 1: Registradores do MIPS suportados pelo simulador.

Registrador	**Descrição**
`zero`	contém a constante zero
`v0, v1`	retorno de funções
`a0, a1, a2, a3`	argumentos
`t0, ... t7`	temporários
`s0, ... s7`	variáveis salvas
`gp`	ponteiro global
`fp`	ponteiro do quadro
`sp`	ponteiro da pilha
`ra`	endereço de retorno
`at, k1, k2`	reservados par ao SO

Tabela 2: Comandos em linguagem de montagem MIPS suportadas pelo simulador.

Categoria	**Nome**	**Sintaxe**	**Significado**
Aritméticas	`add`	`add r1, r2, r3`	`r1 = r2 + r3`
	`addi`	`addi r1, r2, CONST`	`r1 = r2 + CONST`
	`sub`	`sub r1, r2, r3`	`r1 = r2 - r3`
	`mult`	`mult r1, r2`	`lo = ((r1 * r2) « 32) » 32`
			`hi = (r1 * r2) » 32`
	`div`	`div r1, r2`	`lo = r1/r2`
			`hi = r1 % r2`
	`neg`	`neg r1, r2`	`r1 = -r2`
Lógicas	`slt`	`slt r1, r2, r3`	`r1 = (r2 < r3)`
	`slti`	`slti r1, r2, CONST`	`r1 = (r2 < CONST)`
Acesso à Memória	`lw`	`lw r1, CONST(r2)`	`r1 = mem[r2 + CONST]`
	`sw`	`sw r1, CONST(r2)`	`mem[r2 + CONST] = r1`
Desvio Condicional	`beq`	`beq r1, r2, CONST`	`if (r1 == r2)`
			`goto PC + 4 + CONST`
	`bne`	`bne r1, r2, CONST`	`if (r1 != r2)`
			`goto PC + 4 + CONST`
Desvio Incondicional	`j`	`j CONST`	`goto CONST`
	`jr`	`jr r1`	`goto r1`
	`jal`	`jal CONST`	`r31 = PC + 4`
	`jal`	`jal CONST`	`goto CONST`

2. Especificações Técnicas

O programa pode ser implementado em qualquer uma das seguintes linguagens: `C`, `C++`, `Java` ou `Go Lang`. No entanto, a seguinte sintaxe para interface de linha de comando do programa deve ser respeitada:

```
mips32-simulator [nome do arquivo de entrada]
```

O projeto deverá ser obrigatoriamente desenvolvido usando o sistema de versionamento Git. Para hospedar a árvore de fontes, qualquer plataforma de hospedagem de projetos, como o GitHub, BitBucket ou então GitLab, pode ser usada.

Na árvore de fontes do projeto, **informações suficientes para a compilação do programa devem ser fornecidas**. Obrigatoriamente, a compilação deve suportar o ambiente Linux Ubuntu 18.04 e não deve exigir a instalação de pacotes e/ou programas de terceiros (*ie.* IDEs). Portanto, recomenda-se que seja usado um sistema de compilação independente de plataforma, como o `make` ou `cmake` (veja a Seção de Distribuição de Pontos Extras).

3. Distribuição de Pontos

Este trabalho tem o valor de **17 pontos** e deve ser desenvolvido em grupo de dois a quatro integrantes. O link do repositório Git contendo a árvore de fontes do projeto deverá ser entregue em um arquivo texto, que deve ser depositado em uma pasta no SGA antes do prazo para entrega estipulado. *Commits* realizados no repositório após o prazo de entrega no SGA serão desconsiderados. Esse trabalho será avaliado da seguinte forma:

- Corretude da Solução (8 pontos)
- Conformidade com a Especificação (5 pontos)
- Participação de Todos os Integrantes do Grupo (2 pontos)
- Qualidade e Documentação de Código (1 pontos)

A participação de todos dos integrantes do grupo no trabalho será validada caso todos os membros tenham realizado ao menos um *commit* relevante na árvore de fontes. **A ausência de participação de um dos integrantes do grupo implicará em avaliação zero nesse quesito para todos os integrantes**. Discussões entre diferentes grupos da turma são encorajadas. No entanto, qualquer identificação de cópia, implicará na avaliação em zero, para todas as partes.

4. Distribuição de Pontos Extras

Os grupos que desejarem podem realizar uma ou mais das atividades seguintes para obtenção de pontos extras nesse trabalho:

- Automatizar compilação do projeto usando o sistema `make` ou `cmake` (1 ponto).
- Integrar a compilação do projeto comu um ambiente de teste de integração contínuo, como Jenkins ou TravisCI (1 ponto).
- Integrar testes unitários do projeto com um ambiente de teste de integração contínuo, como Jenkins ou TravisCI (1 ponto).

A. Formatos de Arquivo

A.1. Arquivo de Entrada do Simulador

```
add s0, s1, s2
sub s0, s1, s2
mult s0, s1
div s0, s1
and s0, s1, s2
or s0, s1,s2
xor s0, s1, s2
nor s0, s1, s2
slt s0, s1, s2
sll s0, s1, 1
srl s0, s1, 1
jr s0
addi s0, s1, 1
andi s0, s1, 1
ori s0, s1, 1
slti s0, s1, 1
beq s0, s1, 4
bne s0, s1, 4
lw s0, 4(s1)
sw s0, 4(s1)
j 1024
jal 1024
```

Fragmento de Código 1: Exemplo de arquivo de entrada válido para o simulador.

A.2. Arquivos Saída do Simulador

```
00000010001100101000000000100000
00000010001100101000000000100010
00000010000100010000000000011000
00000010000100010000000000011010
00000010001100101000000000100100
00000010001100101000000000100101
00000010001100101000000000100110
00000010001100101000000000100111
00000010001100101000000000101010
00000000000100011000000001000000
00000000000100011000000001000010
00000010000000000000000000001000
00100010001100000000000000000001
00110010001100000000000000000001
00110110001100000000000000000001
00101010001100000000000000000001
00010010001100000000000000000100
00010110001100000000000000000100
10001110001100000000000000000100
10101110001100000000000000000100
00001000000000000000100000000000
00001100000000000000100000000000
```

Fragmento de Código 2: Exemplo do arquivo `instructions.out`.

```
v0 0x00000000
v1 0x00000000
a0 0x00000000
a1 0x00000000
a2 0x00000000
a3 0x00000000
t0 0x00000000
t1 0x00000000
t2 0x00000000
t3 0x00000000
t4 0x00000000
t5 0x00000000
t6 0x00000000
t7 0x00000000
t8 0x00000000
t9 0x00000000
s0 0x00000000
s1 0x00000000
s2 0x00000000
s3 0x00000000
s4 0x00000000
s5 0x00000000
s6 0x00000000
s7 0x00000000
s8 0x00000000
gp 0x00000000
fp 0x00000000
sp 0x00000000
ra 0x00000000
at 0x00000000
k1 0x00000000
k2 0x00000000
```

Fragmento de Código 3: Exemplo do arquivo `register.out`.

```
0x000 0x00000000
0x010 0x00000000
0x020 0x00000000
0x030 0x00000000
0x040 0x00000000
0x050 0x00000000
0x060 0x00000000
0x070 0x00000000
0x080 0x00000000
0x090 0x00000000
0x0a0 0x00000000
0x0b0 0x00000000
0x0c0 0x00000000
0x0d0 0x00000000
0x0e0 0x00000000
0x0f0 0x00000000
0x100 0x00000000
0x110 0x00000000
0x120 0x00000000
0x130 0x00000000
0x140 0x00000000
0x150 0x00000000
0x160 0x00000000
0x170 0x00000000
0x180 0x00000000
0x190 0x00000000
0x1a0 0x00000000
0x1b0 0x00000000
0x1c0 0x00000000
0x1d0 0x00000000
0x1e0 0x00000000
0x1f0 0x00000000
```

Fragmento de Código 4: Exemplo do arquivo `memory.out`.