Trabalho Prático I: Decodificador 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 um processador MIPS. Nesta primeira etapa, você deverá implementar um programa que que decodifica comandos em linguagem de montagem para instruções de máquina do MIPS.

---

1. Descrição

A Tabela 1 detalha a sintaxe e a semântica dos comandos em linguagem de montagem que devem ser suportados pelo decodificador a ser implementado. Em resumo, esse módulo deve ser capaz de decodificar os seguintes comandos em linguagem de montagem para instruções de máquina do MIPS:

- Aritméticas: `add`, `addi` `sub`, `mult`, `div`, `neg`.
- Lógicas: `and`, `andi`, `or`, `ori`, `xor`, `nor`, `slt`, `slti`.
- Deslocamento Bit a Bit: `sll`, `srl`.
- Acesso à Memória: `lw`, `sw`.
- Desvio Condicional: `beq`, `bne`.
- Desvio Incondicional: `j`, `jr`, `jal`.
- Outras: `nop`

Os comandos devem ser lidos de um arquivo de entrada que contém um comando em linguagem de montagem por linha. Por sua vez, cada linha é formada pelo mnemônico da operação, registradores de operando, registrador de resultado e/ou constantes da respectiva instrução. O Fragmento de Código 1 ilustra um exemplo de entrada válida para o programa que deve ser implementado. As instruções de máquina devem ser escritas em um arquivo de saída.

* 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: Comandos em linguagem de montagem MIPS suportadas pelo decodificador.

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	and	and $r1, $r2, $r3	r1 = r2 & r3
	andi	addi $r1, $r2, CONST	r1 = r2 & CONST
	or	or $r1, $r2,$r3	r1 = r2 | r3
	ori	ori $r1, $r2, CONST	r1 = r2 | CONST
	xor	xor $r1, $r2, $r3	r1 = r2 ^ r3
	nor	nor $r1, $r2, $r3	r1 = ~ (r2 | r3)
	slt	slt $r1, $r2, $r3	r1 = (r2 < r3)
	slti	slti $r1, $r2, CONST	r1 = (r2 < CONST)
Deslocamento Bit a Bit	sll	sll $r1, $r2, CONST	r1 = r2 « CONST
	srl	sll $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
Outros	nop	nop	faça nada

```
lw $r1,12 ($r0)
lw $r2,16 ($r0)
lw $r3, 20 ($r0)
lw $r4, 24 ($r0)
mul $r0, $r1, $r2
neg $r5, $r4
div $r5, $r3, $r5
sub $r0, $r0, $r5
sw $r0, 8 ($r0)
```

Fragmento de Código 1: Entrada válida para o decodificador.

2. Especificações Técnicas

O programa decodificador 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-decode [input] [output]`

> Decodifica comandos em linguagens de montagem para instruções de máquina MIPS. Instruções são lidas do arquivo de entrada `input` e escritas no arquivo de saída `output`. Tanto o arquivo de entrada quanto o de saída podem ser omitidos. Nesse caso, comandos em linguagem de montagem devem ser lidos da entrada padrão (*ie.* teclado) e/ou instruções de máquina escritas na saída padrão (*ie.* tela).

O projeto deverá ser necessariamente 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 oito 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 paste no SGA antes do prazo para entrega estipulado. *Commits* realizados no repositório após o prazo de entregue no SGA serão desconsiderados. Esse trabalho será avaliado da seguinte forma:

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

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 e/ou atuado na gestão do projeto, de forma importante (*ie.* criação de *cards*, bugs, *pull requests*, *merges*).

Discussões entre diferentes grupos da turma são encorajadas. No entanto, qualquer identificação de cópia do trabalho, total ou parcial, implicará na avaliação em zero, para ambas 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:

- Esboçar um diagrama de classes do projeto usando uma ferramenta de UML (1 ponto).
- 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).