Operações de Aritmética Simples e Manipulação de Memória Organização de Computadores I

Juliana Miranda Bosio, Vinícios Rosa Buzzi

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 5094 – 88035-972 – Florianópolis – SC – Brazil

Resumo. *Este trabalho apresenta a análise de um código assembly desenvolvido para a arquitetura MIPS, focando em operações aritméticas básicas e manipulação de dados em registradores e memória. O código ilustra o uso de instruções como* `lw`*,* `sw`*,* `addi`*,* `add` *e* `sub` *para realizar cálculos envolvendo variáveis armazenadas. A execução do código foi bem-sucedida, resultando na correta realização das operações aritméticas e no armazenamento do resultado final. A análise ressaltou a importância do entendimento da arquitetura MIPS e da programação assembly para o controle eficiente dos recursos computacionais, além de fornecer uma base sólida para futuras explorações em sistemas de baixo nível.*

Palavras-chave: *Assembly, MIPS, Operações Aritméticas, Instruções lw, sw, addi, add, sub.*

1. Introdução

A linguagem assembly é essencial para a programação de baixo nível, pois permite o controle direto sobre os recursos de hardware, incluindo o gerenciamento de registradores e operações de memória. No contexto da arquitetura MIPS (Microprocessor without Interlocked Pipeline Stages), as operações são realizadas de forma explícita, instrução por instrução, proporcionando uma compreensão clara dos mecanismos subjacentes à execução de programas em nível de máquina.

Este relatório tem como objetivo analisar e descrever o funcionamento de um código assembly escrito para a arquitetura MIPS, que ilustra operações fundamentais de manipulação de dados e memória, como o carregamento de valores em registradores, operações aritméticas e o armazenamento de resultados. Através deste código, serão destacadas instruções importantes, como o uso de lw (load word) para carregar dados da memória para registradores, sw (store word) para armazenar dados em memória, além de operações aritméticas básicas como soma e subtração. A análise visa explorar a interação direta entre registradores e memória, essencial para o desenvolvimento de sistemas de alto desempenho e otimização de recursos computacionais.

2. Objetivo

Estruturar um algoritmo em Assembly e demonstrar, de forma prática, o uso de instruções básicas da arquitetura MIPS para manipulação de registradores e memória, incluindo operações de carga (lw), armazenamento (sw), soma e subtração.

3. Materias e Métodos

Para a execução e análise deste estudo, foi utilizado um código assembly desenvolvido para a arquitetura MIPS (Microprocessor without Interlocked Pipeline Stages). A escolha

dessa arquitetura se deve à sua simplicidade e relevância acadêmica, sendo amplamente utilizada no ensino de organização e arquitetura de computadores. O ambiente de desenvolvimento consistiu em um simulador de MIPS, como o *MARS (MIPS Assembler and Runtime Simulator)*, que permite a simulação de instruções de baixo nível e a análise detalhada do comportamento do processador durante a execução do código.

O código foi composto por dois principais segmentos: o segmento de dados, onde foram declaradas as variáveis, e o segmento de texto, onde foram implementadas as instruções de manipulação dos dados em registradores. As variáveis utilizadas no segmento de dados foram:

- **b**: valor inicializado como -35;
- **d**: valor inicializado como 1;
- **e**: valor inicializado como 0;
- **c**: variável que armazenará o resultado final.

No segmento de texto, foram utilizadas instruções de carga de dados da memória para os registradores, operações aritméticas de soma e subtração, e, por fim, o armazenamento do resultado na memória. As instruções executadas incluíram:

- `lw` (load word): para carregar os valores das variáveis da memória para os registradores;
- `addi` (add immediate): para adicionar um valor imediato ao conteúdo de um registrador;
- `add` (add): para realizar a soma entre dois registradores;
- `sub` (subtract): para subtrair o conteúdo de um registrador de outro;
- `sw` (store word): para armazenar o resultado de um registrador na memória.

De forma analoga, o segundo objetivo do laboratório foi implementar um algoritmo interativo com o usuário, de forma que o mesmo pudesse inserir os valores das variáveis b, d e e, de modo a utilizar as chamadas de sistemas e suas funcionalidades.

O método de execução envolveu a simulação das instruções, verificando o comportamento dos registradores e o armazenamento correto dos resultados. Ao final da execução, o valor resultante foi armazenado na variável `c`. O código foi analisado em termos de seu fluxo lógico e da eficiência na manipulação de registradores e memória, de acordo com os padrões da arquitetura MIPS.

4. Resultados e Discussões

A execução do código assembly desenvolvido para a arquitetura MIPS ocorreu sem problemas, e todos os resultados esperados foram obtidos. Durante o processo, cada instrução foi analisada para garantir que as operações aritméticas e a manipulação de registradores e memória fossem realizadas conforme o planejado.

O fluxo de execução do código pode ser resumido da seguinte forma:

- **1. Carregamento da variável** b: O valor inicial de b (-35) foi carregado com sucesso no registrador $t0$.
- **2. Cálculo de** a: A operação de adição $a = b + 35$ resultou em $a = 0$, o que foi confirmado pelo registrador $t0$ após a execução da instrução `addi`.

- **3. Interação com a variável** e: O valor de e (0) foi carregado e somado ao valor de a, mantendo o resultado como 0.
- **4. Subtração com a variável** d: O valor de d (1) foi subtraído do resultado anterior, resultando em $1 - 0 = 1$. Este valor foi armazenado corretamente na variável c.

Após a execução do código, o valor armazenado na variável c foi verificado, confirmando que o resultado final foi 1, conforme esperado.

Esses resultados demonstram a eficácia das instruções MIPS na manipulação de dados e a interação correta entre registradores e memória. O código conseguiu realizar as operações aritméticas sem erros, evidenciando a importância de uma programação precisa e estruturada em assembly.

De forma semelhante, no processo interativo, observa-se os mesmos deslocamentos e alocações de memória, validando o algoritmo e as chamadas de sistemas. Destaca-se pequena dificuldade de implementação nas chamadas, umas vez que essas precisaram de um pouco mais de compreensão para o entendimento. Ademais, insere que houve uma tentativa de realizar a potenciação utilizando loops e somas, contudo, não houve êxito, uma vez que sempre refletia a necessidade de utilizar instruções ainda não vistas em sala.

Por fim, após a conclusão de ambos os algoritmos se observou os seguintes números de linhas para cada programa.

Table 1. Quantidade de Linhas por Programa

Programa	Linhas Basic	Linhas Source
01	17	12
02	33	29

As diferenças na quantidade de linhas entre o código em `Basic` (alto nível) e `Source` (baixo nível) podem ser atribuídas ao seguinte fator principal:

- **Instruções Explícitas:** Em linguagens de baixo nível, cada operação precisa ser descrita em detalhes, o que exige a utilização de várias instruções para executar algo que em uma linguagem de alto nível poderia ser feito em uma única linha. Por exemplo, em Assembly, operações como somar dois números ou mover dados entre a memória e registradores precisam ser expressas explicitamente com instruções específicas.

Em resumo, a análise e execução deste código assembly ressaltam a relevância do entendimento da arquitetura MIPS para o desenvolvimento de aplicações de baixo nível e para a otimização de recursos computacionais.

5. Conclusões

Este trabalho apresentou uma análise detalhada de um código assembly desenvolvido para a arquitetura MIPS, que realizou operações aritméticas simples envolvendo variáveis armazenadas na memória. Através da implementação prática, foi possível evidenciar a importância das instruções de carga e armazenamento, bem como das operações aritméticas na manipulação de dados.

A execução bem-sucedida do código demonstrou a eficácia da arquitetura MIPS em permitir o controle direto sobre os registradores e a memória, ressaltando a relevância do conhecimento em programação assembly para a compreensão dos princípios de funcionamento dos sistemas computacionais. Os resultados obtidos confirmaram que, com uma estrutura de código adequada, é possível executar operações aritméticas de forma precisa e eficiente.

Além disso, a análise das instruções utilizadas forneceu insights sobre o funcionamento do processador e a interação entre hardware e software em níveis mais profundos. O domínio dessas técnicas é essencial para o desenvolvimento de aplicações que requerem um controle mais fino dos recursos computacionais, especialmente em sistemas embarcados e de alto desempenho.

Por fim, o estudo deste código assembly não apenas reforçou conceitos teóricos de arquitetura de computadores, mas também destacou a importância da prática na formação de profissionais capacitados para lidar com programação de baixo nível, preparando-os para desafios futuros na de ciência da computação.

6. Repositório da Atividade

Confira os códigos implementados acessando o $QRCode$ abaixo ou pelo link:

`https://github.com/buzziologia/UFSC/tree/main/OrganizacaoDeComputadores/Laboratorio01`

Figure 1. Repositório Github