Montador RISC-V
(versão simplificada)

Gabriel Ryan Dos Santos Oliveira - 4688[1], **Gabriel Santos Ferreira de Pádua - 4705**[2]

[1]Universidade Federal de Viçosa (UFV) - Campus Florestal (UFV-CAF)
Florestal – MG – Brasil

{Gabriel,Gabriel}gabriel.oliveira1@ufv.br, gabriel.padua@ufv.br

Abstract. *The practical project consists of implementing a simplified assembler for the RISC-V architecture. The assembler takes as input a file in assembly code format with ".asm" extension and produces as output a file compatible with a previously chosen simulator. The assembler must support a subset of specific instructions. Each group of students is assigned to implement a specific set of instructions, and the assembler must process these instructions according to the assigned set. In short, the project aims to develop low-level language understanding and implementation skills, such as assembly, and familiarize students with the inner workings of RISC-V processors.*

Resumo. *O projeto prático consiste na implementação de um montador simplificado para a arquitetura RISC-V. O montador tem como entrada um arquivo no formato de código assembly com extensão ".asm" e produz como saída um arquivo compatível com um simulador previamente escolhido.O montador deve suportar um subconjunto de instruções específicas. Cada grupo de alunos é designado para implementar um conjunto específico de instruções, e o montador deve processar essas instruções de acordo com o conjunto atribuído. Em suma, o projeto visa desenvolver habilidades de compreensão e implementação de linguagem de baixo nível, como assembly, e familiarizar os alunos com o funcionamento interno de processadores RISC-V.*

1. Informações Gerais

O projeto de implementação do montador RISC-V em linguagem C tem como objetivo principal converter código assembly, seguindo o conjunto de instruções do processador RISC-V, em códigos de máquina binários. O código desenvolvido possibilita o processamento de arquivos de entrada, que devem seguir o padrão de codificação estabelecido e possuir a extensão ".asm". Os arquivos de entrada são recebidos pelo montador através da linha de comando ou podem ser executados diretamente no terminal interativo da aplicação. Os arquivos de entrada tem de seguir o padrão de codificação estabelecido no livro texto, com a extensão ".asm". Este arquivo contém o código assembly a ser convertido pelo montador. O arquivo de saída contém o resultado da montagem e salvo em um arquivo ".txt". Tal arquivo contém as instruções de máquina binárias correspondentes ao código assembly fornecido como entrada. As instruções suportadas incluem LW, SW, SUB, XOR, ADDI, SRL e BEQ, conforme especificado na documentação do projeto. O montador pode ser executado de duas maneiras, recebendo arquivos de entrada pela linha de comando ou executando diretamente no terminal interativo da aplicação.

2. Apresentação do Problema

O trabalho pratico consiste na implementação de uma versão simplificada de um montador RISC-V, este que deve ser capaz suportar um subconjunto instruções predefinas. Deve ser capaz também de reconhecer qual tipo de instrução está sendo passada, e logo após, aplicar a combinação correta a fim de gerar uma sequência binária válida que possa ser interpretada pelo processador.

3. Versionamento do Código

O diretório do trabalho prático é composto por diversos arquivos essenciais para o projeto. Nele, encontra-se os arquivos de código fonte, com extensões ".c" e ".h", onde estão implementadas as funções do projeto. Também estão presentes os arquivos de montagem, com extensão ".asm", responsáveis por armazenar as entradas a serem executadas

guardar as saídas correspondentes aos arquivos de entrada. Por último, temos o "Makefile", que será utilizado para compilar e executar o código de forma conveniente.

4. Linguagem e Compilador

Foi usado a linguagem C para o desenvolvimento do projeto, através do compilador GCC no linux (ubuntu). Comandos:
- sudo apt update
- sudo apt install build-essential
- sudo gcc version

5. Funções

Principais funções usadas no algoritmo para interpreta as instruções e gerar o binário correspondente.

5.1. intToBin

A função "intToBin" é responsável por converte um número inteiro em uma string binária. A conversão é feita utilizando operações bit a bit para obter cada dígito binário da string de saída.

5.2. tipoR

Esta função serve de molde, sendo seus paramentos os sub-pedaços das instruções (opticode ,funct 7, funct 3, rd, rs1, rs2), concatenando e guardando dentro de um vetor de string, no qual servirá para ser exibido, de maneira a mostrar apenas a instrução binaria por completo.

5.3. tipoI

Esta função serve de molde, sendo seus paramentros os sub-pedaços das instruções (opcode ,imediate, funct 3, rd e rs1), concatenando e guardando dentro de um vetor de string, no qual servirá para ser exibido, de maneira a mostrar apenas a instrução binaria por completo.

5.4. montadorBinario

Recebe uma string(instruçoes de entrada) e um ponteiro para outra strg(que sera usada mais tarde) Esta função é responsável por pegar as linhas do arquivo de entrada (instruções), e separar de modo inteligente cada pedaço de instrução (nome da instrução, parte 1, parte 2, parte 3). contendo as partes já separadas, chama: ”montar binário”.

5.5. formatRegs

Função responsável por formatar os registradores e componentes da instrução, de maneira básica, serve para retirar o ”X” , as ”,” os ” ”, ”(” e ”)” das instruções, deixando na string apenas os caracteres que formam nome da instrução (add,ori,...,etc) e os números a serem transformados em binários.

5.6. montarBinario

Recebe uma string (nome da instrução a ser seguida) e pedaços respectivos a essa instrução, usando o nome da instrução, busca assim como deve-se tratar os pedaços no qual recebeu, chamando assim as funções ”tipoR” ou ”TipoI”, adquirindo também um valor para seu funtf 7, funct 3 e opcode de acordo com nome da instrução recebida por paramento.

6. Execução do Código

O projeto tem 2 formas de gerar a saída, sendo por terminal ou por aquivo

6.1. Saída por terminal

Para que a saída seja exibida no terminal pode se executar o comando ”make” e depois um ”make compile” ou simplesmente um ”make all”, e logo apos sera pedido para inserir o nome do aquivo de entrada o qual deseja executar (entrada.asm).

6.2. Saída por aquivo

Para que saída seja armazenada em um aquivo o usuário deve digitar o comando ”make” e logo apos digitar por terminal a entrada convencional de compiladores(./exefile entrada.asm -o saida).

7. Funções Suportadas

O algoritmo reconhece todas as funções passadas em sala de aula até o momento, sendo essas:
Funções:
lb, lh, lw, addi, ori, andi, sb, sh, sw, add, sub, sll, xor, srl, or, and, beq, bne, jalr.

8. Pseudo Instruções

O algoritmo oferece amplitude total em reconhecer todas as pseudos instruções.
Pseudo istructions:
nop, mv, neg, jr rs, jalr rs, ret.

9. Resultados

Resultados exibidos por terminal passando o nome do arquivo ”entrada.asm”:

Resultados exibidos por arquivo passando por linha de comando uma entrada convencional de compiladores

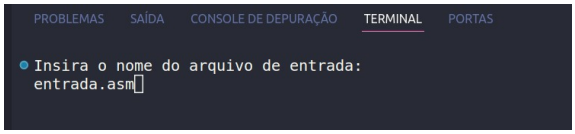

Figure 1. Entrada Terminal

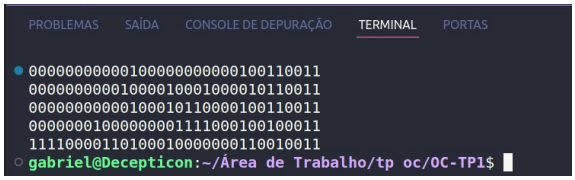

Figure 2. Saída Terminal

Figure 3. Entrada por Linha de Comando

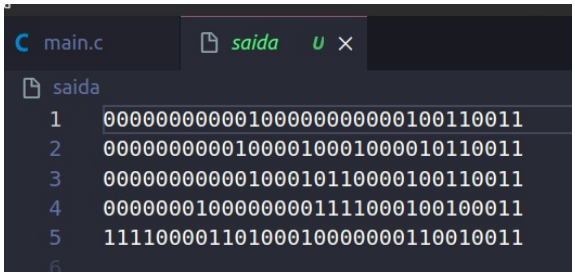

Figure 4. Saída por Linha de Comando

10. Conclusão

Durante o desenvolvimento deste projeto, alcançamos resultados satisfatórios ao atingir nossos objetivos de compilar, interpretar instruções e gerar saídas binárias conforme o esperado. Ao longo da trajetória, encontramos desafios e percalços, porém, com a aplicação de versionamentos e ajustes contínuos no código, conseguimos superá-los de forma eficiente.

Esta experiência nos proporcionou uma compreensão prática da implementação de um montador, permitindo-nos mergulhar profundamente nos detalhes da conversão de código assembly para instruções de máquina. Além disso, fomos capazes de nos familiarizar com o conjunto de instruções do processador RISC-V, explorando suas operações básicas e complexidades.

Ao utilizar a linguagem de programação C, desenvolvemos e aprimoramos nossas habilidades de programação, especialmente em um contexto de baixo nível. A aplicação de conceitos de baixo nível em um ambiente prático nos desafiou a pensar de forma mais detalhada e cuidadosa sobre a manipulação direta da memória e do processador. No geral, este projeto ampliou nosso entendimento sobre os fundamentos da computação e da arquitetura de sistemas. Ao extrair aprendizados valiosos sobre linguagem de baixo nível e a organização interna dos computadores, estamos mais bem preparados para enfrentar desafios futuros na área da ciência da computação.

Portanto, concluímos que este projeto foi uma oportunidade valiosa para aplicar nossos conhecimentos teóricos em um contexto prático, desenvolvendo competências essenciais para nossa formação acadêmica e profissional. A implementação do montador RISC-V em linguagem C ofereceu uma base sólida para nosso entendimento dos princípios de montagem e da arquitetura de processadores, nos preparando para os desafios que encontraremos em nossa jornada futura.

11. GITHUB - Disponibilidade do Código

O código fonte pode ser acessado pelo Github:
- github.com/deyrik/OC-TP1
Obs: branch main sempre será a certa a se seguir

12. Referencias

- The RISC-V Instruction Set Manual - Volume I: User-Level ISA -Document Version 2.2
- Slides do Prof. José Augusto Miranda Nacif
- terminaldeinformacao.com
- pt.stackoverflow.com