Trabalho 2 – OAC

Simulador RISCV:

O intuito do trabalho foi projetar um código em C para executar as funções fetch, decode e execute do RISCV de basic instructions. O intuito do trabalho é entender como funciona a memória do processador e ações de sub-rotinas.

Apresentação do Problema:

O trabalho teve a implementação de um simulador do processador RISCV para códigos já definidos e testar as funções desses códigos a partir de um load word de cada linha da memória, tanto da parte .text quanto da parte .data.

Descrição das instruções:

As funções estão em 2 arquivos: Instructions.c e Memory.c. Todas as funções envolvendo acesso a memória.

As funções do Memory.c se baseiam em ler dados da memória e salvar dados na memória, tanto em word(int32_t), como em inteiro(int8_t) e unsigned (uint8_t) para leitura e word e int8_t para salvar dados.

ao processador estão em Instructions.c, além da lógica de execução de um programa em assembly do RISCV.

fetch(): copia a instrução apontada pelo endereço no registrador program counter (pc) para o registrador ri. Logo após incrementa pc em 4, apontando para a próxima word, que seria a próxima instrução.

decode(): Extrai todos os parâmetros possíveis, simulando como o hardware faria, de forma paralela e calculando todas as possibilidades. Os parâmetros de uma instrução seriam opcode, rs, rt, rd, shamnt, funct, e imm (esse podendo variar de 12 a 20 bits).

execute(): Após ter os parâmetros extraídos da instrução, essa função se baseia neles para fazer as alterações nos registradores. Nessa função utilizamos um switch para opcodes e dentro desse switch, temos outrs switchs para a leitura de todas as unidades possíveis para realização das ações a serem implementadas e caso necessário, realizar alterações na memória, eram elas ações de instruções:

- salto incondicional, salto condicional, relativos ou absolutos e instruções aritméticas e lógicas em geral.
- Além disso implementamos 3 funções de syscall: escrever inteiro, escrever string e finalizar programa.

step(): executa um ciclo de fetch(), decode() e execute().

run(): executa o programa até o fim, considerando fim como o syscall para exit ou alcançar o endereço 0x2000 (fim do segmento de texto).

init(): inicializa os registradores e algumas variáveis de controle do simulador, Além de rodar a memória com os dados do “code.bin” e “data.bin” atuais.

clear_reg(): apaga todo o conteúdo de todos os registradores, incluindo pc, ri, sp e gp. Alterando todos para valor zero.

e lo no formato desejado: d para decimal e h para hexadecimal.

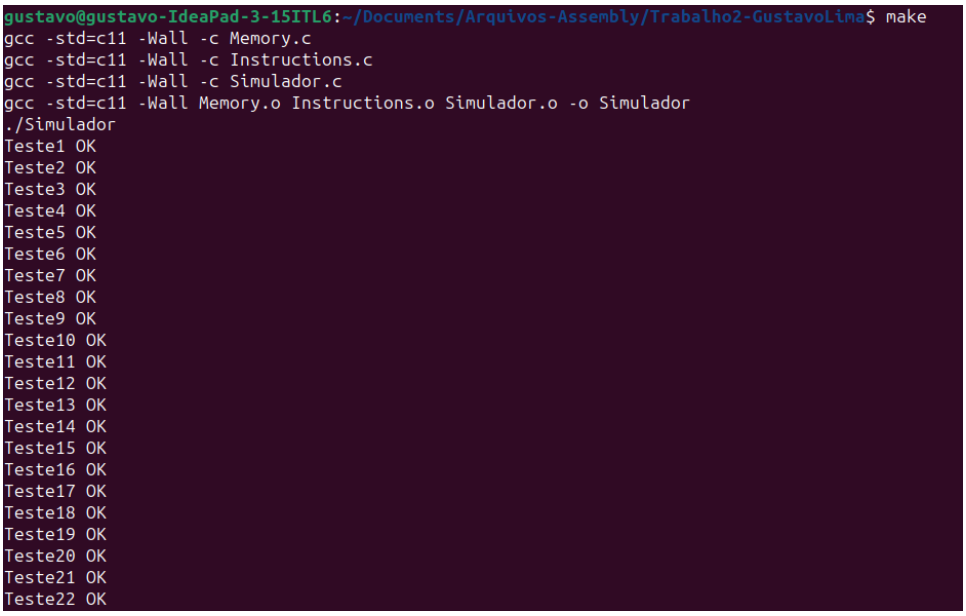
clear_mem(): apaga todo o conteúdo de toda a memória. Alterando todos os dados para valor zero.

load_mem(const char *_fn_, int _start_): abre um arquivo binário para leitura contendo o segmento escolhido (code ou data) e começa a adicionar a partir do ponto de “start” (0 se for code.bin e 0x2000) se for data.bin).

Testes e Resultados:

Os testes feitos foram criados a partir do código dado pelo professor, que testa as instruções de ADD, ADDI, AND, ANDI, AUIPC, BEQ, BNE, BGE, BGEU, BLT, BLTU, LW, LB, LBU, LUI, SLLI, SW, SB, OR, ORI, XOR, SUB, JAL, JALR, SLT, SRA e SRL.

Onde todos os testes foram aprovados, segue a imagem a seguir:



Aluno:

Gustavo Santana Lima

Matrícula:

211038235

Compilador:

GCC (Version 12.3.0)

Sistema Operacional:

Linux Ubuntu 23.04

IDE:

VSCode